edX HarvardX - Data Science: Apex Legends Report

Jason Katsaros

2023-10-06

Contents

1 Introduction

Similar to movies in my previous capstone, Apex Legends is a video game that is very close to my heart. Recently, my brother had the amazing opportunity to pursue a new job in Charlotte, North Carolina, far away from where I currently live. Apex Legends is one way that I get to spend time with my brother, usually on a weekly basis. I always look forward to the time I get to start up a match and just chat with my brother (the game is fun, too). This is why I chose to investigate an aspect of Apex Legends that can possibly be predicted by machine learning. While I know this will not make me better at the game, I still look forward to learning and building something around a special aspect of my life. Through this capstone, I will explain what exactly Apex Legends is, find match data for Apex Legend's main method of gameplay and tidy said data, and create a model to predict the final state of Apex Legends games.

1.1 Motivation

Recently, the company I currently work for put me in an interesting (albeit awkward) position by announcing that I am out of a job by the end of this year. Not one to mope and stand idly by, I took it upon myself to teach myself something new and prepare myself for the next adventure just over the horizon. I find life at its most exciting when I am in the midst of a challenge where I can push through and come out better at the other end. Thus, I decided to participate in the HarvardX: PH125.9x - Data Science program to hone new skills, including but not limited to the R programming language, data science and analysis, and machine learning.

1.2 Overview

Apex Legends("Apex Legends - About" 2019) is a competitive, online first person shooter (FPS) video game developed by Respawn Entertainment and published by Electronic Arts. In Apex Legends, you select a character from a growing cast, each with their own unique passive ability, tactical ability (an ability you can use frequently), and an ultimate ability (a powerful ability you can use infrequently) to beat the other teams (called squads) at some objective, using the vast arsenal of weapons and tools provided to you during each match. While I am not very good at the game yet, I do like to play the character Valkyrie("Apex Legends - Characters" 2019).

While there are many modes of gameplay in Apex (in fact, Respawn Entertainment likes to experiment with different rules, maps, mechanics, etc.), the original mode and focus of this capstone project is the battle royale mode("Apex Legends - Battle Royale" 2019). In battle royale, your squad of three players work together on one big map to eliminate the other nineteen squads (for 60 players total) before they cut you and your teammates out of the game. You can eliminate another player by dealing damage equal to or exceeding their character's health pool plus whatever armor their character might have found and put on during the match. Over the course of the game, a ring closes in on the map, which will deal damage to any player outside of its boundaries, with the purpose of forcing squads closer and closer together, until the ring encompasses everything, putting a final timer on the match. Each match of battle royale has five distinct rings, with the match starting with no ring and ending with the ring covering everything. The five rounds of the ring transition between states at a set period of time within each match and deal different amounts of damage to players to put a timebox on the game. You can see the ring information as shown below("Apex Legends - Endzone & Ring Prediction" 2022):

Round	Time to wait	Time to close	Damage per tick	Ring diameter after closing
1	3 minutes	3 minutes 45 seconds	2	1000 meters
2	2 minutes 45 seconds	45 seconds	3	650 meters
3	2 minutes 15 seconds	45 seconds	10	400 meters
4	1 minute 45 seconds	40 seconds	20	200 meters
5	1 minute 30 seconds	40 seconds	20	100 meters
6	1 minute	2 minutes	25	0.05 meters

In this capstone project, I would like to see if I can use machine learning through R and the `CAReT`(Kuhn and Max 2008), or Classification and Regression Training, package to predict where the final ring will end on a map. While there are also several different maps that put varying points of interest (POI) and obstacles in your path to achieving victory("Apex Legends - Battle Royale - Maps" 2019), this capstone project will only focus on two: World's Edge and Storm Point.

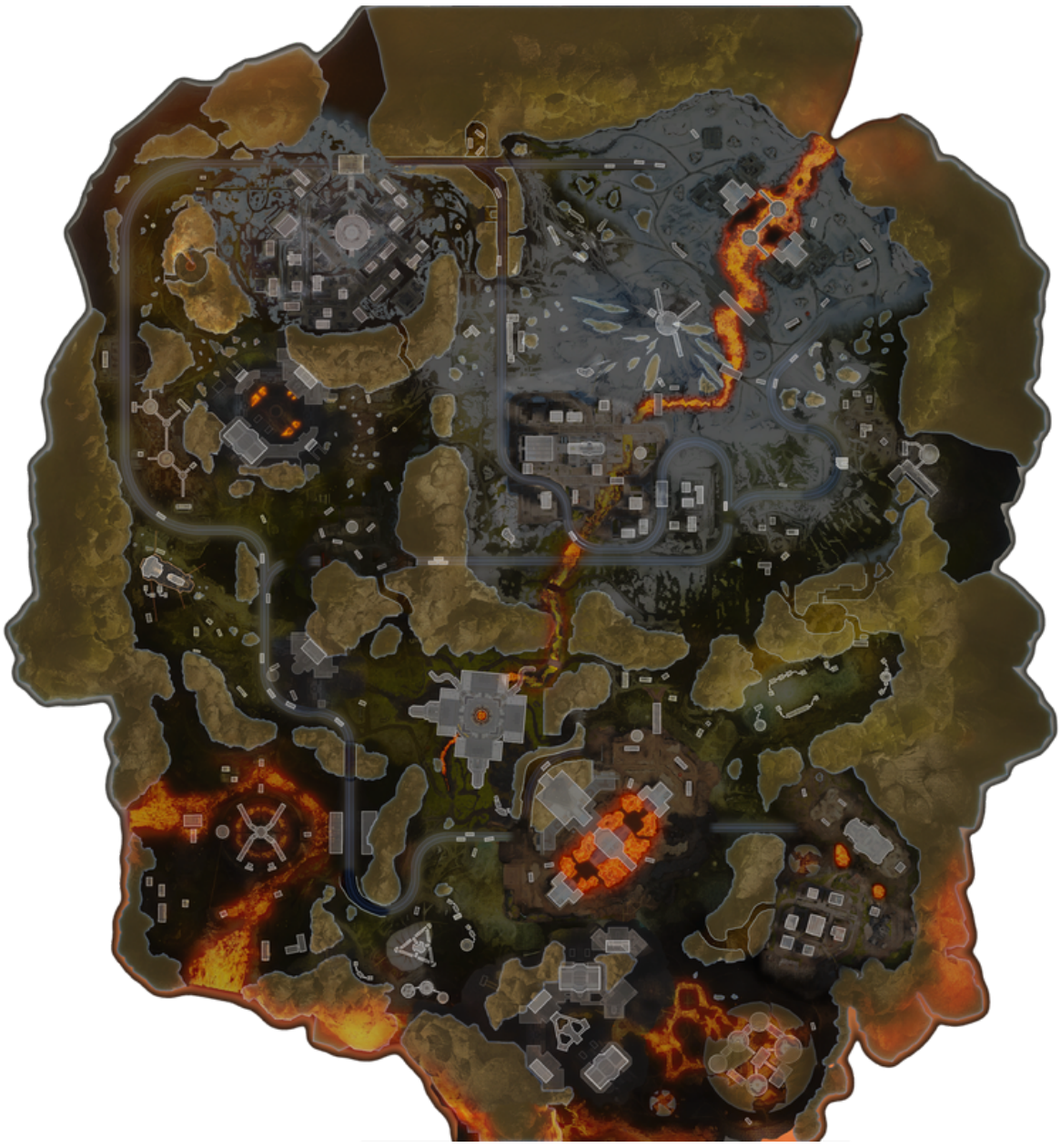


Figure 1: World's Edge Map

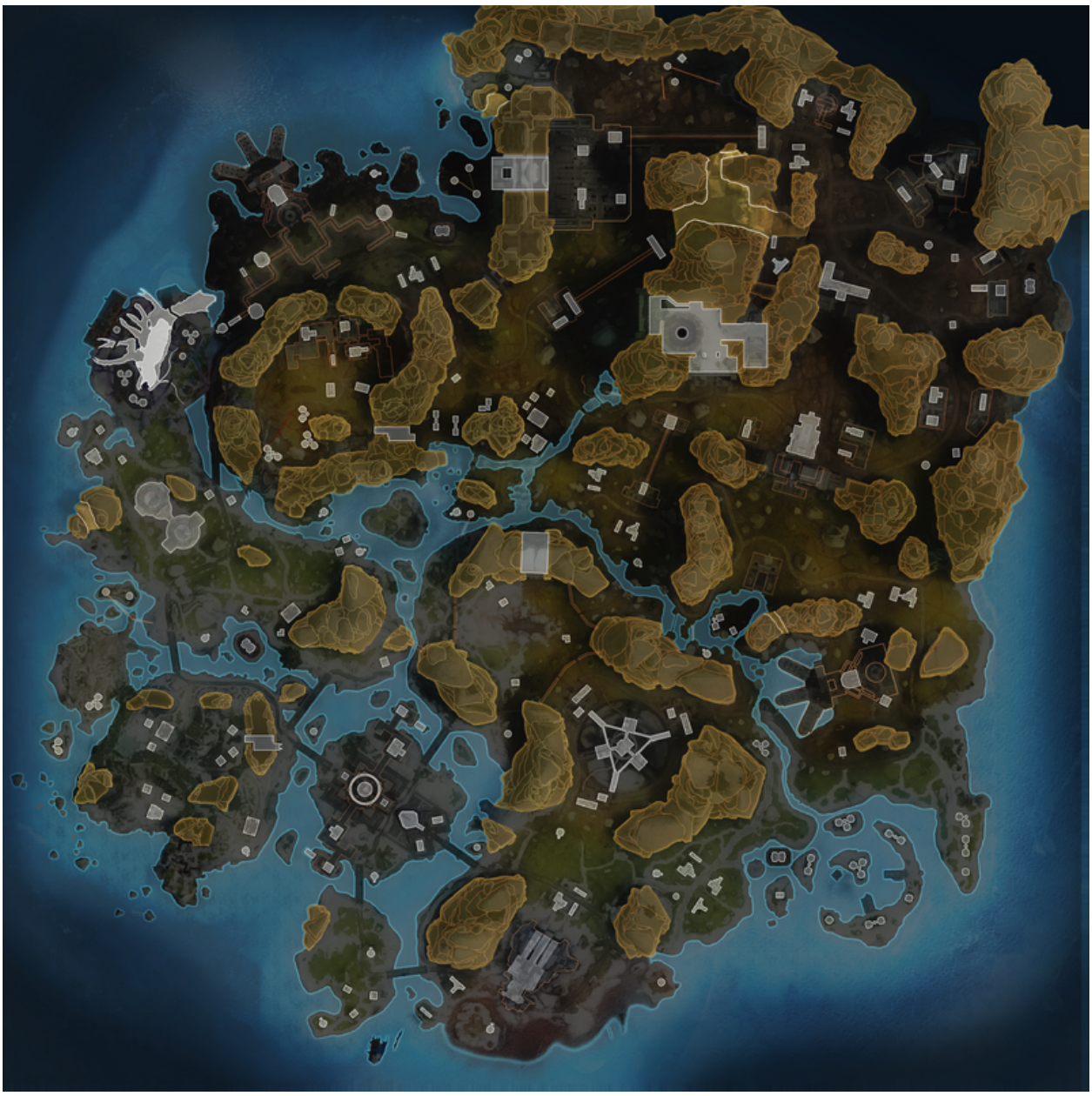


Figure 2: Storm Point Map

2 Data

2.1 Set Up

Throughout this capstone project I will use a number of R packages, including the previously mentioned `caret`(Kuhn and Max 2008) package. I will use the `jsonlite`(Ooms 2014) and the `tidyverse`(Wickham et al. 2019) packages for data manipulation, and the `ggplot2`(Wickham 2016), `ggforce`(Pedersen 2022), `ggimage`(Yu 2023), `gganimate`(Pedersen and Robinson 2022), and `gridExtra`(Auguie 2017) packages for visualizing data.

```
## - The lockfile is already up to date.
```

2.2 Initial Data

To start off this capstone project, I was not given any sort of initial data this time around. Instead, I had to go on the hunt for enough Apex Legends match data that I could create a model using said data. I found a GitHub repository, owned and maintained by the user `bluelightgit`("Apex Zone Predict Machine Learning" 2023) that had Apex Legends match data that fit the bill. The match data in question was stored in JSON (JavaScript Object Notation) format. This is slightly different than the CSV (Comma Separated Values) format that we worked with during the HarvardX Data Science course, but I am used to working with JSON data, being a software developer. The `jsonlite`(Ooms 2014) R package makes it very easy to import JSON data. Using the R code shown below, I download the JSON data and store it in my data directory, import the JSON data into R, then transform the JSON data into a tibble.

```
# Download the json file to the "data" directory
json_file <- "data/apex.json"
if(!file.exists(json_file))
  download.file("https://raw.githubusercontent.com/bluelightgit/apex-zone-predict-machine-learning/main/

# Read and parse the json data
json_data <- fromJSON(json_file)

# Transform the json data into a tibble
apex_data <- as_tibble(json_data)

# Save the apex data to an RData file
save(apex_data, file = "rda/apex_data.rda")

head(apex_data)
```

```
## # A tibble: 6 x 5
##   center$x    $y radius stage gameID                           map
##      <int> <int>  <int> <int> <chr>                            <chr>
## 1     6601 11278  13290     0 09914f6562a97a9dc531f9b141fe482e we
## 2     6601 11278   4369     1 09914f6562a97a9dc531f9b141fe482e we
## 3     4956 12420   2367     2 09914f6562a97a9dc531f9b141fe482e we
## 4     5205 13228   1456     3 09914f6562a97a9dc531f9b141fe482e we
## 5     4920 12846    728     4 09914f6562a97a9dc531f9b141fe482e we
## 6     4751 12686    364     5 09914f6562a97a9dc531f9b141fe482e we
```

2.3 Pre-Processing

Before I continue to analyzing the data, I am first going to perform some pre-processing on the Apex Legends match data. Throughout the course of this capstone project, I am going to work with the match data represented in two different ways:

1. A long format, which is easier to use when visualizing data, and

2. A wide format, which is easier to use when training and evaluating the model I am going to create.

2.3.1 Long Data

Firstly, I want to tidy up the Apex Legends match data, which will make viewing the data but also make creating the wide data much easier in the long run. In the long representation of the Apex Legends match data, each row will correspond to a stage of the ring, meaning each game will have five rows.

1. Currently, the x and y coordinates of each stage of the ring within each game in the data set are nested within their own data frame. To make the x and y coordinates easier to access, I want to bring them out from their own nested data frame and into each row.
2. The `map` column is not very human-readable at the moment, with `we` corresponding to `World's Edge` and `sp` corresponding to `Storm Point`. I want to convert the `map` column into a factor and also change the name of said factors to be very clear what map each row matches with.
3. Speaking of factors, I will also make the `stage` and `gameID` columns factors.

```
# Copy the apex data to a new variable for manipulation
apex_data_tidy <- apex_data
# Bring the x and y coordinates of the ring out from a nested data frame
# and into their own columns in the data set
apex_data_tidy <- apex_data_tidy %>%
  mutate(x = center$x, y = center$y) %>%
  select(-center)
# Rename "we" to "World's Edge" and "sp" to "Storm Point" respectively
# Make the "map" column values factors
apex_data_tidy <- apex_data_tidy %>%
  mutate(map = as.factor(ifelse(map == "we", "World's Edge", "Storm Point")))
# Make the "stage" and "gameID" column values factors
apex_data_tidy <- apex_data_tidy %>%
  mutate(stage = as.factor(stage), gameID = as.factor(gameID))

# Save the tidy apex data to an RData file
save(apex_data_tidy, file = "rda/apex_data_tidy.rda")
```

After cleaning up the Apex Legends match data a little bit, there is also some information that I would like to calculate and represent in the data, which I believe will be helpful later on when creating the model for predicting the final ring state in a match.

1. The distance between each ring stage, calculated like so:

$$\sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$$

where i represents the ring stages, 1 through 5 (0 being the ring stage where there is no ring on the map).

2. The angle between each ring stage, calculated like so:

$$\arctan m_i$$

where m is the slope of the line drawn between the ring stages and i represents the ring stages, 1 through 5 (0 being the ring stage where there is no ring on the map).

```
# Copy the tidy apex data to a new variable for further manipulation
apex_data_long <- apex_data_tidy

# Create and populate a new variable with 0
distance_between <- rep(0.0, nrow(apex_data_long))
```

```r
# For each ring (and the ring to follow),
# if the current ring is not either
# 1. the ring that does not display (ring 0)
# 2. or the last ring of the match (ring 5)
# then calculate the distance between the two ring stages
for (current_ring in 1:nrow(apex_data_long)) {
  if (apex_data_long[current_ring,]$stage != 0 | apex_data_long[current_ring,]$stage != 5) {
    distance_between[current_ring] <- sqrt((apex_data_long[current_ring + 1,]$x - apex_data_long[curren
  }
}

# Transform any NaN into 0
distance_between[is.nan(distance_between)] <- 0

# Add the new information to the apex_data_long data set
apex_data_long <- apex_data_long %>%
  mutate(distance_between = distance_between)

# Create and populate a new variable with 0
angle_between <- rep(0.0, nrow(apex_data_long))

# For each ring (and the ring to follow),
# if the current ring is not either
# 1. the ring that does not display (ring 0)
# 2. or the last ring of the match (ring 5)
# then calculate the angle between the two ring stages
for (current_ring in 1:nrow(apex_data_long)) {
  if (apex_data_long[current_ring,]$stage != 0 | apex_data_long[current_ring,]$stage != 5) {
    angle_between[current_ring] <- atan((apex_data_long[current_ring + 1,]$y - apex_data_long[current_ri
  }
}

# Transform any NaN into 0
angle_between[is.nan(angle_between)] <- 0

# Add the new information to the apex_data_long data set
apex_data_long <- apex_data_long %>%
  mutate(angle_between = angle_between)

# Save the long apex data to an RData file
save(apex_data_long, file = "rda/apex_data_long.rda")
```

2.3.2 Wide Data

Next, I want to create a wide representation of the Apex Legends match data, which I will use when training and evaluating the model for predicting the final ring location on a map. To do this, I will mutate the data set with similar information to the long representation of the data, but I will also make use of the `pivot_wider` function.

```r
# Helper function to account for any NaN encountered within a data.frame
is.nan.data.frame <- function(df)
  do.call(cbind, lapply(df, is.nan))

# Copy the tidy apex data to a new variable for further manipulation
```

```r
apex_data_wide <- apex_data_tidy
# Make a wide representation of the data set
# by making each row represent one whole game
# with all ring stage coordinates represented by their own columns
apex_data_wide <- apex_data_wide %>%
  filter(map == "World's Edge") %>%
  pivot_wider(
    names_from = stage,
    values_from = c(x, y, radius)
  ) %>%
  mutate(
    distance_from_center_1 = sqrt((x_1 - mean(apex_data_wide$x))^2 + (y_1 - mean(apex_data_wide$y))^2),
    distance_between_1_and_2 = sqrt((x_2 - x_1)^2 + (y_2 - y_1)^2),
    angle_between_1_and_2 = atan((y_2 - y_1) / (x_2 - x_1)),
    distance_from_center_2 = sqrt((x_2 - mean(apex_data_wide$x))^2 + (y_2 - mean(apex_data_wide$y))^2),
    distance_between_2_and_3 = sqrt((x_3 - x_2)^2 + (y_3 - y_2)^2),
    angle_between_2_and_3 = atan((y_3 - y_2) / (x_3 - x_2)),
    distance_from_center_3 = sqrt((x_3 - mean(apex_data_wide$x))^2 + (y_3 - mean(apex_data_wide$y))^2),
    distance_between_3_and_4 = sqrt((x_4 - x_3)^2 + (y_4 - y_3)^2),
    angle_between_3_and_4 = atan((y_4 - y_3) / (x_4 - x_3)),
    distance_from_center_4 = sqrt((x_4 - mean(apex_data_wide$x))^2 + (y_4 - mean(apex_data_wide$y))^2),
    distance_between_4_and_5 = sqrt((x_5 - x_4)^2 + (y_5 - y_4)^2),
    angle_between_4_and_5 = atan((y_5 - y_4) / (x_5 - x_4)),
    distance_from_center_5 = sqrt((x_5 - mean(apex_data_wide$x))^2 + (y_5 - mean(apex_data_wide$y))^2),
  )

apex_data_wide[is.nan(apex_data_wide)] <- 0

# Save the wide apex data to an RData file
save(apex_data_wide, file = "rda/apex_data_wide.rda")
```

Now each row in this data set represents one game as a whole, with columns corresponding to each x and y coordinate of each stage of the ring.

2.4 Visualizing Apex Legends Matches

Before I continue with the analysis portion of this capstone project, I would like to take a moment to look at the data I have transformed to see if I can notice any trends or tidbits of information that I might find useful later on when I am conducting machine learning.

```r
# Summarize the radius data across rings on each map
apex_data_long %>%
  group_by(map, stage) %>% # Each map is different from the next, so only compare rings within the same
  summarize(
    radius_average = mean(radius),
    radius_standard_deviation = sd(radius),
    radius_min = min(radius),
    radius_max = max(radius)
  )
```

```
## # A tibble: 12 x 6
## # Groups:   map [2]
##    map         stage radius_average radius_standard_devi~1 radius_min radius_max
##    <fct>       <fct>          <dbl>                  <dbl>      <int>      <int>
##  1 Storm Point 0             12675.                  0.517      12672      12675
```

```
##  2 Storm Point 1              4492.                  0.495       4491       4493
##  3 Storm Point 2              2406.                  0.509       2405       2407
##  4 Storm Point 3              1283.                  0.462       1283       1284
##  5 Storm Point 4               642.                  0.271        641        642
##  6 Storm Point 5               321                   0            321        321
##  7 World's Ed~ 0             13283.                 67.2        12646      13291
##  8 World's Ed~ 1              4369.                  0.544       4368       4370
##  9 World's Ed~ 2              2367.                  0.479       2366       2367
## 10 World's Ed~ 3              1456.                  0.381       1456       1457
## 11 World's Ed~ 4               728                   0            728        728
## 12 World's Ed~ 5               364                   0            364        364
## # i abbreviated name: 1: radius_standard_deviation
```

```
# Summarize the ring area data across rings on each map
apex_data_long %>%
  group_by(map, stage) %>% # Each map is different from the next, so only compare rings within the same
  summarize(
    area_average = mean(pi * radius^2),
    area_standard_deviation = sd(pi * radius^2),
    area_min = min(pi * radius^2),
    area_max = max(pi * radius^2)
  )
```

```
## # A tibble: 12 x 6
## # Groups:   map [2]
##    map          stage area_average area_standard_deviation   area_min   area_max
##    <fct>        <fct>        <dbl>                   <dbl>      <dbl>      <dbl>
##  1 Storm Point  0       504696636.                  41157. 504475641. 504714531.
##  2 Storm Point  1        63393161.                  13982.  63363037.  63419485.
##  3 Storm Point  2        18191771.                   7690.  18171050.  18201285.
##  4 Storm Point  3         5173788.                   3728.   5171341.   5179406.
##  5 Storm Point  4         1294534.                   1091.   1290821.   1294851.
##  6 Storm Point  5          323713.                      0     323713.    323713.
##  7 World's Edge 0       554322415.                5472508. 502407631. 554964482.
##  8 World's Edge 1        59968122.                  14926.  59939778.  59994681.
##  9 World's Edge 2        17596195.                   7121.  17586497.  17601367.
## 10 World's Edge 3         6661567.                   3488.   6659975.   6669127.
## 11 World's Edge 4         1664994.                      0    1664994.   1664994.
## 12 World's Edge 5          416248.                      0     416248.    416248.
```

```
# Summarize the distance between rings data across rings on each map
apex_data_long %>%
  group_by(map, stage) %>% # Each map is different from the next, so only compare rings within the same
  summarize(
    distance_between_average = mean(distance_between),
    distance_between_standard_deviation = sd(distance_between),
    distance_between_min = min(distance_between),
    distance_between_max = max(distance_between)
  )
```

```
## # A tibble: 12 x 6
## # Groups:   map [2]
##    map          stage distance_between_average distance_between_standard_devia~1
##    <fct>        <fct>                    <dbl>                              <dbl>
##  1 Storm Point  0                       0.0608                              0.254
##  2 Storm Point  1                    2065.                                 79.2
```

```
##  3 Storm Point  2                    1061.                                183.
##  4 Storm Point  3                     474.                                210.
##  5 Storm Point  4                     260.                                 88.0
##  6 Storm Point  5                      NA                                  NA
##  7 World's Edge 0                       0.102                               0.378
##  8 World's Edge 1                    1964.                                109.
##  9 World's Edge 2                     781.                                166.
## 10 World's Edge 3                     553.                                185.
## 11 World's Edge 4                     239.                                102.
## 12 World's Edge 5                    5192.                               2173.
## # i abbreviated name: 1: distance_between_standard_deviation
## # i 2 more variables: distance_between_min <dbl>, distance_between_max <dbl>
```

```
# Summarize the angle between rings data across rings on each map
apex_data_long %>%
  group_by(map, stage) %>% # Each map is different from the next, so only compare rings within the same
  summarize(
    angle_between_average = mean(angle_between),
    angle_between_standard_deviation = sd(angle_between),
    angle_between_min = min(angle_between),
    angle_between_max = max(angle_between)
  )
```

```
## # A tibble: 12 x 6
## # Groups:   map [2]
##    map      stage angle_between_average angle_between_standa~1 angle_between_min
##    <fct>    <fct>                 <dbl>                  <dbl>             <dbl>
##  1 Storm P~ 0                  -0.0265                  0.301             -1.57
##  2 Storm P~ 1                   0.0770                  1.00              -1.56
##  3 Storm P~ 2                  -0.0492                  0.878             -1.52
##  4 Storm P~ 3                  -0.168                   0.931             -1.56
##  5 Storm P~ 4                   0.0694                  0.955             -1.55
##  6 Storm P~ 5                  NA                      NA                 NA
##  7 World's~ 0                   0.00854                 0.339             -1.57
##  8 World's~ 1                   0.0671                  0.949             -1.55
##  9 World's~ 2                   0.0805                  0.884             -1.55
## 10 World's~ 3                   0.00847                 0.891             -1.54
## 11 World's~ 4                  -0.174                   0.927             -1.55
## 12 World's~ 5                  -0.0144                  0.898             -1.56
## # i abbreviated name: 1: angle_between_standard_deviation
## # i 1 more variable: angle_between_max <dbl>
```

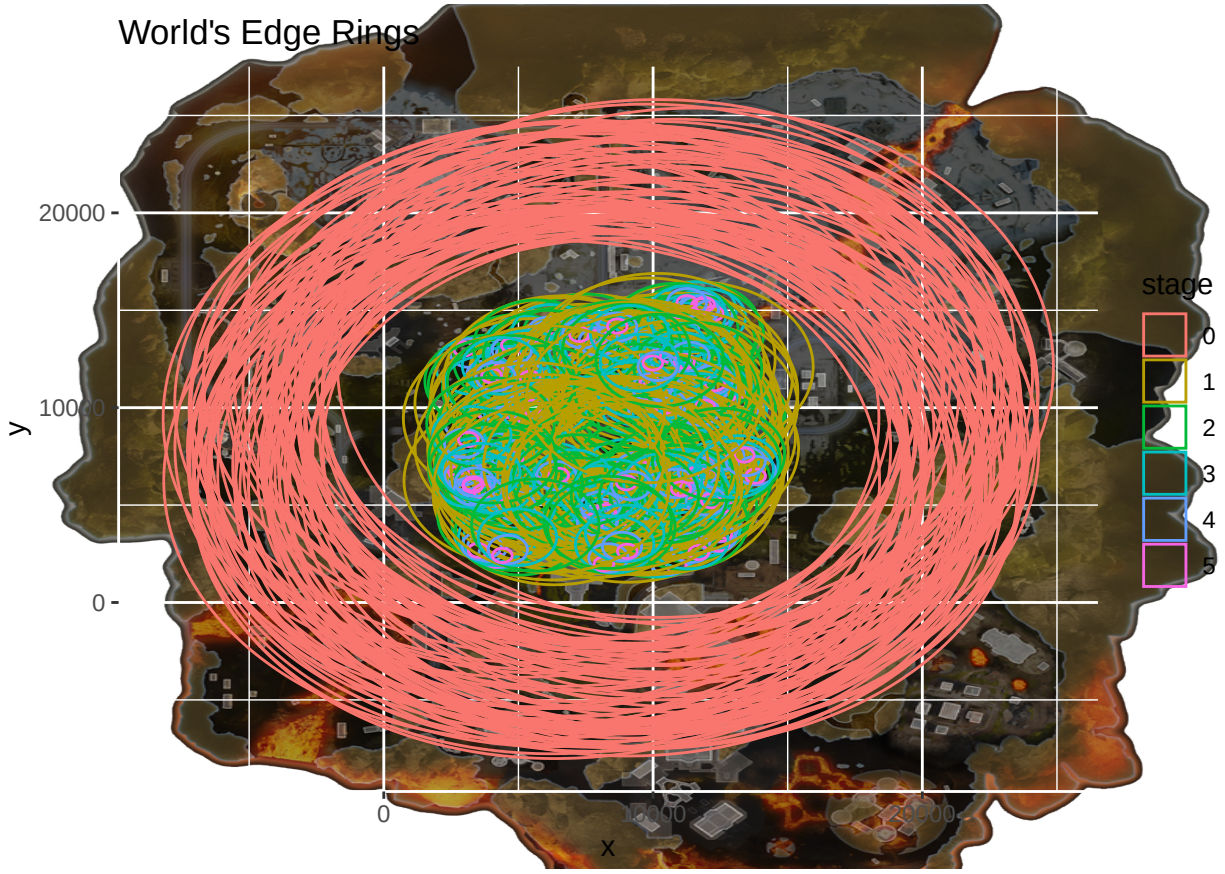
After summarizing the data, we can now see that:

1. The size of each stage of the ring are consistent across games on each map
2. The distance between each stage of the ring is mostly consistent across games on each map
3. The angle between each stage of the ring is also mostly consistent across games on each map

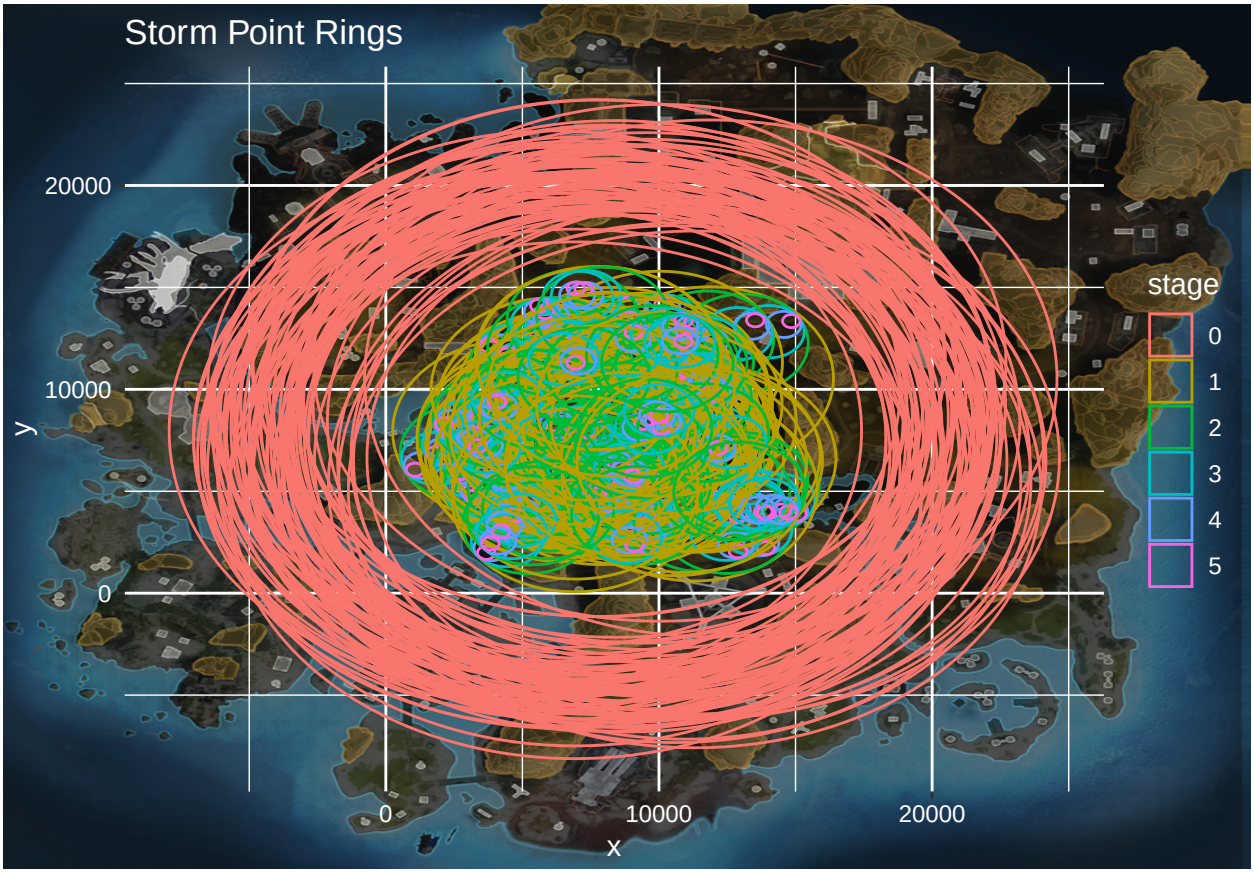
These three facts mean that I should be able to predict the final ring location in a game fairly accurately. Below are two plots superimposed on top of an image of each map to give the general idea of where each stage of the ring travels on each map.

```
# Visualize ring movement across all games on World's Edge
# Animated plot not shown in the report as gganimate doesn't work with knitr
worlds_edge_plot <- apex_data_long %>%
  filter(map == "World's Edge") %>%
  ggplot(aes(x0 = x, y0 = y, r = radius, group = gameID, color = stage)) +
```

```
  geom_circle() +
  ggtitle("World's Edge Rings") +
  theme(
    axis.line = element_line(color = "white")
  )
ggbackground(worlds_edge_plot, worlds_edge_image_file)
```




```
# Visualize ring movement across all games on Storm Point
# Animated plot not shown in the report as gganimate doesn't work with knitr
storm_point_plot <- apex_data_long %>%
  filter(map == "Storm Point") %>%
  ggplot(aes(x0 = x, y0 = y, r = radius, group = gameID, color = stage)) +
  geom_circle() +
  ggtitle("Storm Point Rings") +
  theme(
    axis.text = element_text(color = "white"),
    legend.text = element_text(color = "white"),
    title = element_text(color = "white")
  )
ggbackground(storm_point_plot, storm_point_image_file)
```



As you can see in the plots above, the ring tends to draw players towards the center of each map (which isn't always the case, but happens to be how each of these games turned out in the data set). The final ring demonstrates where highly contested POI are located on each map. Since it seems like I can anticipate the trajectory of each ring stage, this seems to be a good topic to train a machine learning model on to see how accurately I can predict the location of the final ring of an Apex Legends match.

2.5 Train and Test Sets

Since I am trying to predict two variables, the x and the y coordinate of the center of the final ring in an Apex Legends battle royale match, I will split the data into separate x and y train and test sets. I will also create two models and two sets of predictions later on.

```
# Create training and testing partitions of the wide apex data
# These sets are separate from the y train and test data sets
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
# set.seed(1) # if using R 3.5 or earlier
x_test_index <- createDataPartition(apex_data_wide$x_5, times = 1, p = 0.5, list = FALSE)
x_train_set <- apex_data_wide[-x_test_index,]
x_test_set <- apex_data_wide[x_test_index,]

# Create training and testing partitions of the wide apex data
# These sets are separate from the x train and test data sets
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
# set.seed(1) # if using R 3.5 or earlier
y_test_index <- createDataPartition(apex_data_wide$y_5, times = 1, p = 0.5, list = FALSE)
y_train_set <- apex_data_wide[-y_test_index,]
y_test_set <- apex_data_wide[y_test_index,]
```

After conducting several test runs with the `caret`(Kuhn and Max 2008) package, I decided on a 50/50 split between the train and test sets. This combination produced the best RMSE against the `glm` method, which I will discuss further below.

3 Analysis

3.1 A Note On Accuracy

Because predicting the exact x and y coordinates out of these large maps will be incredibly difficult (if not impossible) for my models to do, I will evaluate the accuracy of my model's predictions on a sliding scale. I am interested to know, which predictions were within 250 meters? 100 meters? 50 meters? 25 meters? In the grand scheme of an Apex Legends match, these distances are relatively small, and this measure of accuracy will help determine whether to follow my model's predictions or not while playing a match of Apex Legends. Being in the general vicinity of the final ring long before the final ring closes in is an incredibly strategic move, and could make the difference between clinching the win or losing unprepared.

I will also use the Root Mean Squared Error (or RMSE for short) to help evaluate the fit of my model towards the problem at hand. RMSE is a loss function used to evaluate how well an algorithm predicts outcomes. Creating a model that minimizes the RMSE indicates that the model is making accurate predictions. Mathematically, RMSE is defined as:

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}$$

where $\hat{y}$ is a prediction, y is an observed outcome, N is the number of observations made, and i is the current prediction/observation in the summation. We can calculate the RMSE with the following R code:

```
RMSE <- function(observed, prediction) {
  sqrt(mean((observed - prediction)^2))
}
```

The `caret`(Kuhn and Max 2008) package also contains a function to calculate the RMSE:

```
?caret::RMSE()
```

In this case, however, since I am already using the `caret`(Kuhn and Max 2008) package to create the models (unlike in my MovieLens capstone), the model object already contains the RMSE information, which means there is no need to calculate it separately.

3.2 Generalized Linear Model

The first method I am going to use to train my models to predict the location of the final ring in an Apex Legends match will be the "Generalized Linear Model" method, or `glm` for short. `glm` is essentially linear regression covered in the HarvardX Data Science course previously as well as in my other capstone project, the MovieLens recommendation analysis with the main difference being that the "Generalized Linear Model" can fit more complex models than just regular linear regression. To see more information about this method, you can execute the R code seen below.

```
getModelInfo("glm")
```

This method has no tuning parameters to tweak how the algorithm is run in the back-end. However, this should give me a good baseline with which I can continue my investigation. In the following R code, I will train two models separately, one for predicting x coordinates and another for predicting y coordinates. I will then use these two separate models to make distinct predictions for x and y coordinates. After these models make their predictions, I will evaluate their predictions' accuracy based on the sliding scale discussed above against the actual x and y coordinates of the final ring location in the corresponding games.

```r
# Generalized Linear Model
# Train a model to predict the x coordinate of the final stage of the ring in an Apex Legends match
x_model <- train(
  x_5 ~ x_0 + x_1 + x_2 + x_3 + x_4, # Predict on the other 4 x coordinates of the other ring stages
  data = x_train_set,
  method = "glm"
)

# Make predictions for x_5 in the x_test_set based on the x_model
x_predictions <- predict(x_model, newdata = x_test_set)

# Since I'm not looking to be exact, evaluate the accuracy of the predictions on a sliding scale
glm_x_accuracy_within_250_meters <- mean(x_predictions <= x_test_set$x_5 + 250 & x_predictions >= x_tes
glm_x_accuracy_within_100_meters <- mean(x_predictions <= x_test_set$x_5 + 100 & x_predictions >= x_tes
glm_x_accuracy_within_50_meters <- mean(x_predictions <= x_test_set$x_5 + 50 & x_predictions >= x_test_
glm_x_accuracy_within_25_meters <- mean(x_predictions <= x_test_set$x_5 + 25 & x_predictions >= x_test_

# Train a model to predict the y coordinate of the final stage of the ring in an Apex Legends match
y_model <- train(
  y_5 ~ y_0 + y_1 + y_2 + y_3 + y_4, # Predict on the other 4 y coordinates of the other ring stages
  data = y_train_set,
  method = "glm"
)

# Make predictions for y_5 in the y_test_set based on the y_model
y_predictions <- predict(y_model, newdata = y_test_set)

# Since I'm not looking to be exact, evaluate the accuracy of the predictions on a sliding scale
glm_y_accuracy_within_250_meters <- mean(y_predictions <= y_test_set$y_5 + 250 & y_predictions >= y_tes
glm_y_accuracy_within_100_meters <- mean(y_predictions <= y_test_set$y_5 + 100 & y_predictions >= y_tes
glm_y_accuracy_within_50_meters <- mean(y_predictions <= y_test_set$y_5 + 50 & y_predictions >= y_test_
glm_y_accuracy_within_25_meters <- mean(y_predictions <= y_test_set$y_5 + 25 & y_predictions >= y_test_

# Print and save the accuracy results to a results variable
accuracy_results <- data.frame(
  method = "glm",
  x_250 = glm_x_accuracy_within_250_meters,
  x_100 = glm_x_accuracy_within_100_meters,
  x_50 = glm_x_accuracy_within_50_meters,
  x_25 = glm_x_accuracy_within_25_meters,
  y_250 = glm_y_accuracy_within_250_meters,
  y_100 = glm_y_accuracy_within_100_meters,
  y_50 = glm_y_accuracy_within_50_meters,
  y_25 = glm_y_accuracy_within_25_meters
)
accuracy_results %>% knitr::kable()
```

method	x_250	x_100	x_50	x_25	y_250	y_100	y_50	y_25
glm	0.875	0.3958333	0.2083333	0.1666667	0.7708333	0.2916667	0.0833333	0.0416667

```r
# Save the results to a RData file
save(accuracy_results, file = "rda/accuracy_results.rda")
```

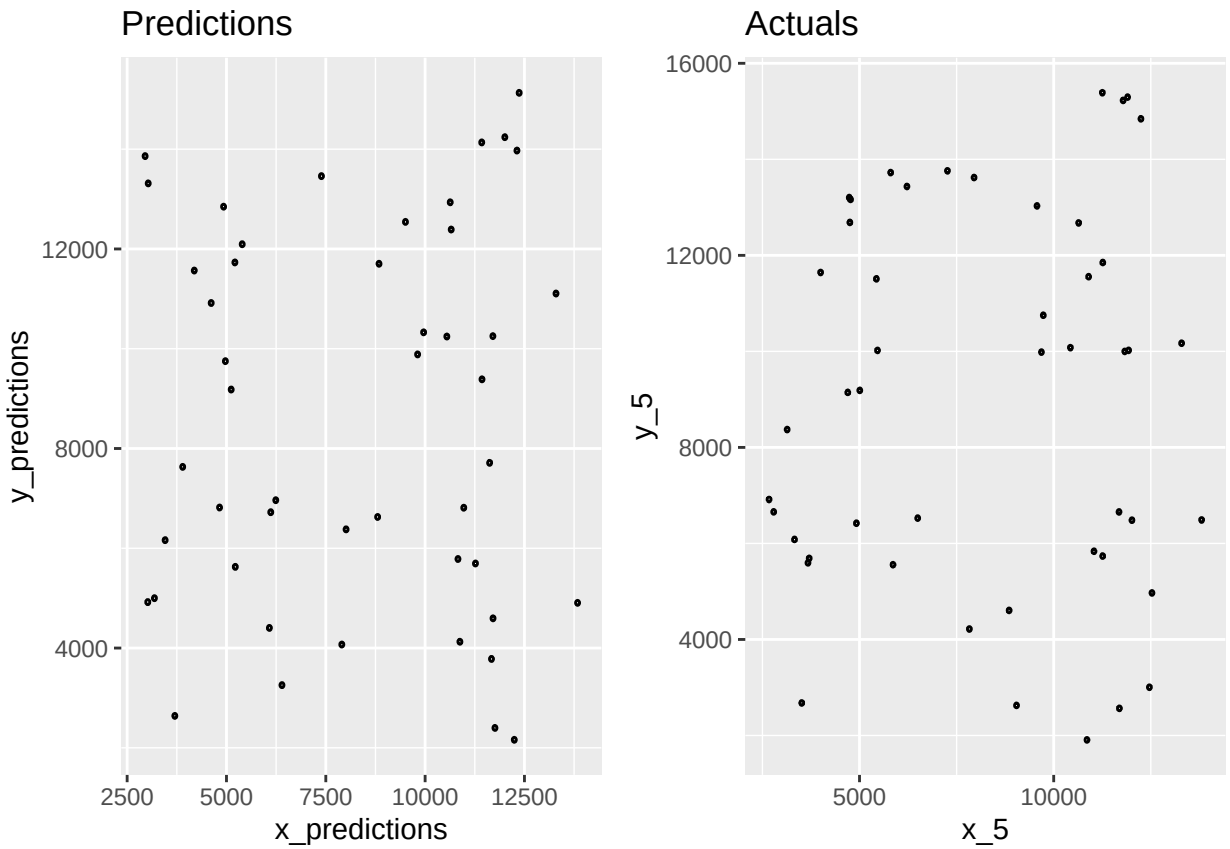
```r
rmse_results <- data.frame(
  method = "glm",
  x_RMSE = x_model$results$RMSE,
  y_RMSE = y_model$results$RMSE
)
rmse_results %>% knitr::kable()
```

method	x_RMSE	y_RMSE
glm	222.6889	230.7225

```r
# Save the results to a RData file
save(rmse_results, file = "rda/rmse_results.rda")
```

As we can see, these basic models have an accuracy in the upper 80s for within 250 meters, give or take, which I would say is very good. Movement characters (like Valkyrie) can easily reposition your squad hundreds of meters at a time, so these predictions will get us pretty much right where we need to be. The RMSEs is not as good as our MovieLens capstone project RMSEs, but because we're not looking for pinpoint accuracy in this project, the RMSEs here are still in a good spot.

```r
# Set the ring diameter to 100
ring_5_diameter <- 100
# Plot a circle the size of the final ring for each x and y coordinate prediction
predictions_plot <- data.frame(x_predictions, y_predictions) %>%
  ggplot(aes(x0 = x_predictions, y0 = y_predictions, r = ring_5_diameter / 2)) +
  geom_circle() +
  ggtitle("Predictions")
# Plot a circle the size of the final ring for each actual x and y coordinate
actuals_plot <- apex_data_wide %>%
  filter(gameID %in% x_test_set$gameID) %>%
  ggplot(aes(x0 = x_5, y0 = y_5, r = ring_5_diameter / 2)) +
  geom_circle() +
  ggtitle("Actuals")
# Display the two plots side-by-side
grid.arrange(predictions_plot, actuals_plot, ncol = 2)
```



The plots above show the locations of the predicted final rings, with the x and y coordinates now combined, against the actual final ring locations for the games in the test sets. While the two plots may look dissimilar, the general shape and locations of the predicted final rings follow the trend set forth by the actual final ring locations, which is a good sign.

3.3 Generalized Linear Model With Stepwise Feature Selection

Taking the "Generalized Linear Model" to the next level, here I decided to try out the "Generalized Linear Model With Stepwise Feature Selection". You can view more information about this method by executing the R code shown below.

```
getModelInfo("glmStepAIC")
```

This method is similar to the "Generalized Linear Model" method, except that it automatically chooses the optimal set of predictors through an iterative testing process. With the verbose output enabled, you can see the different combinations of supplied predictors this algorithm creates and tries out.

```
# Generalized Linear Model With Stepwise Feature Selection
# Train a model to predict the x coordinate of the final stage of the ring in an Apex Legends match
x_model <- train(
  x_5 ~ x_0 + x_1 + x_2 + x_3 + x_4, # Predict on the other 4 x coordinates of the other ring stages
  data = x_train_set,
  method = "glmStepAIC",
  verbose = FALSE # suppress verbose output
)
```

```
## Start:  AIC=594.43
## .outcome ~ x_0 + x_1 + x_2 + x_3 + x_4
```

```
##
##
## Step:  AIC=594.43
## .outcome ~ x_0 + x_2 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_3   1  1445701 592.46
## - x_2   1  1453886 592.71
## - x_0   1  1478857 593.46
## <none>     1444683 594.43
## - x_4   1  5903316 654.37
##
## Step:  AIC=592.46
## .outcome ~ x_0 + x_2 + x_4
##
##        Df Deviance    AIC
## - x_2   1  1453931 590.71
## - x_0   1  1482913 591.58
## <none>     1445701 592.46
## - x_4   1 16930799 698.73
##
## Step:  AIC=590.71
## .outcome ~ x_0 + x_4
##
##        Df Deviance    AIC
## - x_0   1  1484372 589.62
## <none>     1453931 590.71
## - x_4   1 66050636 756.62
##
## Step:  AIC=589.62
## .outcome ~ x_4
##
##        Df  Deviance    AIC
## <none>      1484372 589.62
## - x_4   1 467831522 840.76
## Start:  AIC=579.92
## .outcome ~ x_0 + x_1 + x_2 + x_3 + x_4
##
##        Df Deviance    AIC
## <none>      992664 579.92
## - x_1   1  1080493 581.65
## - x_0   1  1080549 581.65
## - x_3   1  1151987 584.47
## - x_2   1  1327447 590.71
## - x_4   1  6984436 663.77
## Start:  AIC=582.59
## .outcome ~ x_0 + x_1 + x_2 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_2   1  1054902 580.60
## - x_3   1  1075264 581.44
## <none>     1054760 582.59
## - x_1   1  1135661 583.84
## - x_0   1  1135712 583.84
```

```
## - x_4   1  5390750 652.37
##
## Step:  AIC=580.6
## .outcome ~ x_0 + x_1 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_3   1  1079979 579.63
## <none>     1054902 580.60
## - x_1   1  1137335 581.91
## - x_0   1  1137387 581.91
## - x_4   1  5408962 650.52
##
## Step:  AIC=579.63
## .outcome ~ x_0 + x_1 + x_4
##
##        Df Deviance    AIC
## <none>     1079979 579.63
## - x_1   1  1161443 580.83
## - x_0   1  1161500 580.83
## - x_4   1 46909957 743.57
## Start:  AIC=588.05
## .outcome ~ x_0 + x_1 + x_2 + x_3 + x_4
##
##        Df Deviance    AIC
## <none>     1194117 588.05
## - x_3   1  1266118 588.63
## - x_1   1  1356619 591.66
## - x_0   1  1356714 591.67
## - x_2   1  1412507 593.44
## - x_4   1  6168372 658.30
## Start:  AIC=588.29
## .outcome ~ x_0 + x_1 + x_2 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_2   1  1201441 586.32
## - x_3   1  1220549 587.01
## - x_1   1  1235665 587.56
## - x_0   1  1235670 587.56
## <none>     1200761 588.29
## - x_4   1  6690983 661.88
##
## Step:  AIC=586.32
## .outcome ~ x_0 + x_1 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_3   1  1222602 585.09
## - x_1   1  1236951 585.60
## - x_0   1  1236957 585.60
## <none>     1201441 586.32
## - x_4   1  6833066 660.80
##
## Step:  AIC=585.09
## .outcome ~ x_0 + x_1 + x_4
##
```

```
##        Df Deviance    AIC
## - x_1   1  1258174 584.35
## - x_0   1  1258178 584.35
## <none>     1222602 585.09
## - x_4   1 52539103 748.55
##
## Step:  AIC=584.35
## .outcome ~ x_0 + x_4
##
##        Df Deviance    AIC
## - x_0   1  1259020 582.38
## <none>     1258174 584.35
## - x_4   1 61191349 753.26
##
## Step:  AIC=582.38
## .outcome ~ x_4
##
##        Df  Deviance    AIC
## <none>      1259020 582.38
## - x_4   1 434693596 837.53
## Start:  AIC=583.66
## .outcome ~ x_0 + x_1 + x_2 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_2   1  1081000 581.67
## - x_1   1  1086780 581.91
## - x_0   1  1086785 581.91
## - x_3   1  1090697 582.06
## <none>     1080731 583.66
## - x_4   1  7469645 666.72
##
## Step:  AIC=581.67
## .outcome ~ x_0 + x_1 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_1   1  1086783 579.91
## - x_0   1  1086789 579.91
## - x_3   1  1097180 580.33
## <none>     1081000 581.67
## - x_4   1  7469735 664.72
##
## Step:  AIC=579.91
## .outcome ~ x_0 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_3   1  1099767 578.43
## - x_0   1  1112990 578.95
## <none>     1086783 579.91
## - x_4   1  7480036 662.78
##
## Step:  AIC=578.43
## .outcome ~ x_0 + x_4
##
##        Df Deviance    AIC
```

```
## - x_0   1  1123313 577.36
## <none>     1099767 578.43
## - x_4   1 77520330 763.67
##
## Step:  AIC=577.36
## .outcome ~ x_4
##
##        Df  Deviance    AIC
## <none>      1123313 577.36
## - x_4   1 371884762 830.66
## Start:  AIC=586.29
## .outcome ~ x_0 + x_1 + x_2 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_3   1  1169864 585.15
## - x_2   1  1178826 585.48
## - x_1   1  1179902 585.52
## - x_0   1  1179941 585.53
## <none>     1147378 586.29
## - x_4   1  8589675 672.87
##
## Step:  AIC=585.15
## .outcome ~ x_0 + x_1 + x_2 + x_4
##
##        Df Deviance    AIC
## - x_2   1  1180555 583.55
## - x_1   1  1198562 584.21
## - x_0   1  1198597 584.22
## <none>     1169864 585.15
## - x_4   1 18135115 703.75
##
## Step:  AIC=583.55
## .outcome ~ x_0 + x_1 + x_4
##
##        Df Deviance    AIC
## - x_1   1  1206831 582.52
## - x_0   1  1206864 582.52
## <none>     1180555 583.55
## - x_4   1 82216627 768.26
##
## Step:  AIC=582.52
## .outcome ~ x_0 + x_4
##
##        Df Deviance    AIC
## <none>     1206831 582.52
## - x_0   1  1361219 585.81
## - x_4   1 92930478 771.65
## Start:  AIC=593.86
## .outcome ~ x_0 + x_1 + x_2 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_2   1  1374804 592.25
## - x_3   1  1376397 592.30
## - x_1   1  1381095 592.45
```

```
## - x_0   1  1381104 592.45
## <none>     1362769 593.86
## - x_4   1  6037590 657.36
##
## Step:  AIC=592.25
## .outcome ~ x_0 + x_1 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_3   1  1381570 590.47
## - x_1   1  1400010 591.05
## - x_0   1  1400018 591.05
## <none>     1374804 592.25
## - x_4   1  6387587 657.84
##
## Step:  AIC=590.47
## .outcome ~ x_0 + x_1 + x_4
##
##        Df Deviance    AIC
## - x_1   1  1401980 589.11
## - x_0   1  1401987 589.11
## <none>     1381570 590.47
## - x_4   1 52424308 748.46
##
## Step:  AIC=589.11
## .outcome ~ x_0 + x_4
##
##        Df Deviance    AIC
## - x_0   1  1411380 587.41
## <none>     1401980 589.11
## - x_4   1 78706591 764.34
##
## Step:  AIC=587.41
## .outcome ~ x_4
##
##        Df  Deviance    AIC
## <none>      1411380 587.41
## - x_4   1 521571795 845.55
## Start:  AIC=592.04
## .outcome ~ x_0 + x_1 + x_2 + x_3 + x_4
##
##
## Step:  AIC=592.04
## .outcome ~ x_0 + x_2 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_2   1  1368834 590.06
## - x_3   1  1376543 590.31
## - x_0   1  1408726 591.32
## <none>     1368160 592.04
## - x_4   1  5654300 652.47
##
## Step:  AIC=590.06
## .outcome ~ x_0 + x_3 + x_4
##
```

```
##        Df Deviance    AIC
## - x_3   1  1376868 588.32
## - x_0   1  1408864 589.33
## <none>     1368834 590.06
## - x_4   1  5678663 650.66
##
## Step:  AIC=588.32
## .outcome ~ x_0 + x_4
##
##        Df Deviance    AIC
## - x_0   1  1411719 587.42
## <none>     1376868 588.32
## - x_4   1 79678422 764.88
##
## Step:  AIC=587.42
## .outcome ~ x_4
##
##        Df  Deviance    AIC
## <none>      1411719 587.42
## - x_4   1 400866984 833.96
## Start:  AIC=593.65
## .outcome ~ x_0 + x_1 + x_2 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_1   1  1358766 591.73
## - x_0   1  1358780 591.73
## <none>     1356135 593.65
## - x_3   1  1425729 593.85
## - x_2   1  1438714 594.25
## - x_4   1  6638682 661.53
##
## Step:  AIC=591.73
## .outcome ~ x_0 + x_2 + x_3 + x_4
##
##        Df Deviance    AIC
## <none>     1358766 591.73
## - x_3   1  1428027 591.92
## - x_2   1  1441461 592.33
## - x_0   1  1550691 595.55
## - x_4   1  6694367 659.90
## Start:  AIC=596.11
## .outcome ~ x_0 + x_1 + x_2 + x_3 + x_4
##
##
## Step:  AIC=596.11
## .outcome ~ x_0 + x_2 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_0   1  1502093 594.15
## - x_3   1  1502245 594.15
## - x_2   1  1557530 595.74
## <none>     1500922 596.11
## - x_4   1  5921733 654.50
##
```

```
## Step:  AIC=594.15
## .outcome ~ x_2 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_3   1  1503971 592.20
## - x_2   1  1559195 593.79
## <none>     1502093 594.15
## - x_4   1  7170260 660.92
##
## Step:  AIC=592.2
## .outcome ~ x_2 + x_4
##
##        Df Deviance    AIC
## - x_2   1  1566120 591.98
## <none>     1503971 592.20
## - x_4   1 12494500 683.36
##
## Step:  AIC=591.98
## .outcome ~ x_4
##
##        Df  Deviance    AIC
## <none>      1566120 591.98
## - x_4   1 505674294 844.18
## Start:  AIC=584.77
## .outcome ~ x_0 + x_1 + x_2 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_3   1  1108368 582.77
## - x_1   1  1109946 582.83
## - x_0   1  1109952 582.83
## - x_2   1  1119259 583.20
## <none>     1108240 584.77
## - x_4   1  7539946 667.13
##
## Step:  AIC=582.77
## .outcome ~ x_0 + x_1 + x_2 + x_4
##
##        Df Deviance    AIC
## - x_1   1  1110094 580.84
## - x_0   1  1110101 580.84
## - x_2   1  1125003 581.43
## <none>     1108368 582.77
## - x_4   1 11977968 685.50
##
## Step:  AIC=580.84
## .outcome ~ x_0 + x_2 + x_4
##
##        Df Deviance    AIC
## - x_2   1  1129440 579.60
## <none>     1110094 580.84
## - x_0   1  1216957 582.88
## - x_4   1 12773035 686.33
##
## Step:  AIC=579.6
```

```
## .outcome ~ x_0 + x_4
##
##        Df Deviance    AIC
## <none>     1129440 579.60
## - x_0   1  1220762 581.02
## - x_4   1 67384170 757.50
## Start:  AIC=593.82
## .outcome ~ x_0 + x_1 + x_2 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_3   1  1362738 591.86
## - x_2   1  1390479 592.75
## - x_1   1  1395594 592.91
## - x_0   1  1395595 592.91
## <none>     1361353 593.82
## - x_4   1  6351266 659.59
##
## Step:  AIC=591.86
## .outcome ~ x_0 + x_1 + x_2 + x_4
##
##        Df Deviance    AIC
## - x_2   1  1392480 590.81
## - x_0   1  1399248 591.03
## - x_1   1  1399249 591.03
## <none>     1362738 591.86
## - x_4   1 13518264 690.82
##
## Step:  AIC=590.81
## .outcome ~ x_0 + x_1 + x_4
##
##        Df Deviance    AIC
## - x_0   1  1418000 589.61
## - x_1   1  1418006 589.61
## <none>     1392480 590.81
## - x_4   1 49749109 746.15
##
## Step:  AIC=589.61
## .outcome ~ x_1 + x_4
##
##        Df Deviance    AIC
## - x_1   1  1423513 587.78
## <none>     1418000 589.61
## - x_4   1 51896535 746.01
##
## Step:  AIC=587.78
## .outcome ~ x_4
##
##        Df  Deviance    AIC
## <none>      1423513 587.78
## - x_4   1 482035789 842.08
## Start:  AIC=595.35
## .outcome ~ x_0 + x_1 + x_2 + x_3 + x_4
##
##        Df Deviance    AIC
```

```
## - x_1   1  1413605 593.48
## - x_0   1  1413612 593.48
## - x_2   1  1414661 593.51
## - x_3   1  1430810 594.01
## <none>     1409598 595.35
## - x_4   1  6608856 661.33
##
## Step:  AIC=593.48
## .outcome ~ x_0 + x_2 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_2   1  1417057 591.58
## - x_0   1  1429154 591.96
## - x_3   1  1439883 592.29
## <none>     1413605 593.48
## - x_4   1  7831367 666.80
##
## Step:  AIC=591.58
## .outcome ~ x_0 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_0   1  1429484 589.97
## - x_3   1  1440120 590.29
## <none>     1417057 591.58
## - x_4   1  7843189 664.87
##
## Step:  AIC=589.97
## .outcome ~ x_3 + x_4
##
##        Df Deviance    AIC
## - x_3   1  1462289 588.96
## <none>     1429484 589.97
## - x_4   1  7870210 663.02
##
## Step:  AIC=588.96
## .outcome ~ x_4
##
##        Df  Deviance    AIC
## <none>      1462289 588.96
## - x_4   1 494218586 843.18
## Start:  AIC=591.07
## .outcome ~ x_0 + x_1 + x_2 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_1   1  1280028 589.11
## - x_0   1  1280031 589.11
## - x_3   1  1286367 589.32
## <none>     1278995 591.07
## - x_2   1  1339881 591.12
## - x_4   1  5813458 655.69
##
## Step:  AIC=589.11
## .outcome ~ x_0 + x_2 + x_3 + x_4
##
```

```
##        Df Deviance    AIC
## - x_3   1  1286457 587.33
## - x_0   1  1332375 588.87
## <none>     1280028 589.11
## - x_2   1  1341184 589.16
## - x_4   1  6022816 655.25
##
## Step:  AIC=587.33
## .outcome ~ x_0 + x_2 + x_4
##
##        Df Deviance    AIC
## - x_0   1  1342050 587.19
## <none>     1286457 587.33
## - x_2   1  1347301 587.36
## - x_4   1 15320216 694.33
##
## Step:  AIC=587.19
## .outcome ~ x_2 + x_4
##
##        Df Deviance    AIC
## - x_2   1  1377178 586.33
## <none>     1342050 587.19
## - x_4   1 15417368 692.61
##
## Step:  AIC=586.33
## .outcome ~ x_4
##
##        Df  Deviance    AIC
## <none>      1377178 586.33
## - x_4   1 423924234 836.43
## Start:  AIC=577.34
## .outcome ~ x_0 + x_1 + x_2 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_3   1   941316 575.58
## - x_2   1   954693 576.20
## <none>      936025 577.34
## - x_1   1   992383 577.91
## - x_0   1   992446 577.91
## - x_4   1  6811781 662.67
##
## Step:  AIC=575.58
## .outcome ~ x_0 + x_1 + x_2 + x_4
##
##        Df Deviance    AIC
## - x_2   1   954696 574.20
## <none>      941316 575.58
## - x_1   1   994539 576.00
## - x_0   1   994601 576.01
## - x_4   1 16405863 699.34
##
## Step:  AIC=574.2
## .outcome ~ x_0 + x_1 + x_4
##
```

```
##        Df Deviance    AIC
## <none>      954696 574.20
## - x_1   1  1034319 575.73
## - x_0   1  1034406 575.73
## - x_4   1 87664641 771.08
## Start:  AIC=585.83
## .outcome ~ x_0 + x_1 + x_2 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_2   1  1136367 583.87
## - x_3   1  1149942 584.39
## <none>     1135219 585.83
## - x_1   1  1235976 587.57
## - x_0   1  1235987 587.57
## - x_4   1  7258594 665.46
##
## Step:  AIC=583.87
## .outcome ~ x_0 + x_1 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_3   1  1150330 582.41
## <none>     1136367 583.87
## - x_1   1  1236949 585.60
## - x_0   1  1236957 585.60
## - x_4   1  7815066 666.71
##
## Step:  AIC=582.41
## .outcome ~ x_0 + x_1 + x_4
##
##        Df Deviance    AIC
## <none>     1150330 582.41
## - x_1   1  1250169 584.07
## - x_0   1  1250183 584.07
## - x_4   1 53512264 749.36
## Start:  AIC=596.3
## .outcome ~ x_0 + x_1 + x_2 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_1   1  1468362 595.15
## - x_0   1  1468365 595.15
## - x_3   1  1475068 595.35
## - x_2   1  1497711 596.02
## <none>     1440204 596.30
## - x_4   1  7740486 668.29
##
## Step:  AIC=595.15
## .outcome ~ x_0 + x_2 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_0   1  1469476 593.18
## - x_3   1  1506514 594.28
## - x_2   1  1518255 594.62
## <none>     1468362 595.15
## - x_4   1  8149360 668.55
```

```
##
## Step:  AIC=593.18
## .outcome ~ x_2 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_3   1  1507637 592.31
## - x_2   1  1518318 592.62
## <none>     1469476 593.18
## - x_4   1  8432371 668.06
##
## Step:  AIC=592.31
## .outcome ~ x_2 + x_4
##
##        Df Deviance    AIC
## - x_2   1  1523492 590.77
## <none>     1507637 592.31
## - x_4   1 17323656 697.74
##
## Step:  AIC=590.77
## .outcome ~ x_4
##
##        Df  Deviance    AIC
## <none>      1523492 590.77
## - x_4   1 502008648 843.86
## Start:  AIC=590.72
## .outcome ~ x_0 + x_1 + x_2 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_3   1  1279721 589.10
## - x_1   1  1286090 589.32
## - x_0   1  1286136 589.32
## - x_2   1  1302513 589.87
## <none>     1268814 590.72
## - x_4   1  8450980 672.15
##
## Step:  AIC=589.1
## .outcome ~ x_0 + x_1 + x_2 + x_4
##
##        Df Deviance    AIC
## - x_1   1  1302464 587.87
## - x_0   1  1302512 587.87
## - x_2   1  1302764 587.88
## <none>     1279721 589.10
## - x_4   1 17685863 702.65
##
## Step:  AIC=587.87
## .outcome ~ x_0 + x_2 + x_4
##
##        Df Deviance    AIC
## - x_2   1  1329900 586.79
## <none>     1302464 587.87
## - x_0   1  1459100 590.87
## - x_4   1 19284945 704.46
##
```

```
## Step:  AIC=586.79
## .outcome ~ x_0 + x_4
##
##        Df Deviance    AIC
## <none>     1329900 586.79
## - x_0   1  1459379 588.88
## - x_4   1 97044527 773.55
## Start:  AIC=599.93
## .outcome ~ x_0 + x_1 + x_2 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_3   1  1566003 597.98
## - x_2   1  1601340 598.96
## - x_1   1  1613756 599.30
## - x_0   1  1613766 599.30
## <none>     1564185 599.93
## - x_4   1  6142131 658.11
##
## Step:  AIC=597.98
## .outcome ~ x_0 + x_1 + x_2 + x_4
##
##        Df Deviance    AIC
## - x_2   1  1609334 597.18
## - x_1   1  1618837 597.44
## - x_0   1  1618850 597.44
## <none>     1566003 597.98
## - x_4   1 14265965 693.19
##
## Step:  AIC=597.18
## .outcome ~ x_0 + x_1 + x_4
##
##        Df Deviance    AIC
## - x_1   1  1647362 596.21
## - x_0   1  1647369 596.21
## <none>     1609334 597.18
## - x_4   1 70001139 761.18
##
## Step:  AIC=596.21
## .outcome ~ x_0 + x_4
##
##        Df Deviance    AIC
## - x_0   1  1653439 594.37
## <none>     1647362 596.21
## - x_4   1 71565923 760.15
##
## Step:  AIC=594.37
## .outcome ~ x_4
##
##        Df  Deviance    AIC
## <none>      1653439 594.37
## - x_4   1 548584092 847.77
## Start:  AIC=592.77
## .outcome ~ x_0 + x_1 + x_2 + x_3 + x_4
##
```

```
##
## Step:  AIC=592.77
## .outcome ~ x_0 + x_2 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_0   1  1391298 590.78
## - x_3   1  1414569 591.51
## - x_2   1  1417629 591.60
## <none>     1391123 592.77
## - x_4   1  6701194 659.95
##
## Step:  AIC=590.78
## .outcome ~ x_2 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_3   1  1415358 589.53
## - x_2   1  1420016 589.67
## <none>     1391298 590.78
## - x_4   1  6867019 659.02
##
## Step:  AIC=589.53
## .outcome ~ x_2 + x_4
##
##        Df Deviance    AIC
## - x_2   1  1425466 587.84
## <none>     1415358 589.53
## - x_4   1 15079304 691.63
##
## Step:  AIC=587.84
## .outcome ~ x_4
##
##        Df  Deviance    AIC
## <none>      1425466 587.84
## - x_4   1 476275398 841.55
## Start:  AIC=590
## .outcome ~ x_0 + x_1 + x_2 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_2   1  1250944 588.10
## - x_1   1  1254169 588.21
## - x_0   1  1254171 588.21
## - x_3   1  1258245 588.35
## <none>     1248246 590.00
## - x_4   1  7131362 664.68
##
## Step:  AIC=588.1
## .outcome ~ x_0 + x_1 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_1   1  1258996 586.38
## - x_0   1  1259000 586.38
## - x_3   1  1273344 586.88
## <none>     1250944 588.10
## - x_4   1  7131362 662.68
```

```
##
## Step:  AIC=586.38
## .outcome ~ x_0 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_0   1  1263379 584.53
## - x_3   1  1291236 585.49
## <none>     1258996 586.38
## - x_4   1  7940026 665.41
##
## Step:  AIC=584.53
## .outcome ~ x_3 + x_4
##
##        Df Deviance    AIC
## - x_3   1  1303527 583.91
## <none>     1263379 584.53
## - x_4   1  7955375 663.49
##
## Step:  AIC=583.91
## .outcome ~ x_4
##
##        Df  Deviance    AIC
## <none>      1303527 583.91
## - x_4   1 531674008 846.39
## Start:  AIC=580.76
## .outcome ~ x_0 + x_1 + x_2 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_0   1  1017958 579.03
## - x_1   1  1017981 579.03
## - x_2   1  1020444 579.14
## - x_3   1  1029761 579.54
## <none>     1011837 580.76
## - x_4   1  7278450 665.58
##
## Step:  AIC=579.03
## .outcome ~ x_1 + x_2 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_2   1  1025056 577.33
## - x_3   1  1033049 577.68
## <none>     1017958 579.03
## - x_1   1  1159330 582.75
## - x_4   1  7844023 666.87
##
## Step:  AIC=577.33
## .outcome ~ x_1 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_3   1  1058033 576.73
## <none>     1025056 577.33
## - x_1   1  1230685 583.38
## - x_4   1  7852134 664.92
##
```

```
## Step:  AIC=576.73
## .outcome ~ x_1 + x_4
##
##        Df Deviance    AIC
## <none>     1058033 576.73
## - x_1   1  1268188 582.70
## - x_4   1 50861765 745.13
## Start:  AIC=590.29
## .outcome ~ x_0 + x_1 + x_2 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_3   1  1260329 588.43
## - x_2   1  1262259 588.49
## - x_1   1  1272518 588.85
## - x_0   1  1272547 588.85
## <none>     1256591 590.29
## - x_4   1  8571923 672.78
##
## Step:  AIC=588.43
## .outcome ~ x_0 + x_1 + x_2 + x_4
##
##        Df Deviance    AIC
## - x_2   1  1262485 586.50
## - x_1   1  1276740 586.99
## - x_0   1  1276769 587.00
## <none>     1260329 588.43
## - x_4   1 20196119 708.49
##
## Step:  AIC=586.5
## .outcome ~ x_0 + x_1 + x_4
##
##        Df Deviance    AIC
## - x_1   1  1279235 585.08
## - x_0   1  1279263 585.08
## <none>     1262485 586.50
## - x_4   1 41257235 737.92
##
## Step:  AIC=585.08
## .outcome ~ x_0 + x_4
##
##        Df Deviance    AIC
## <none>     1279235 585.08
## - x_0   1  1356304 585.65
## - x_4   1 56011402 749.37
## Start:  AIC=589.78
## .outcome ~ x_0 + x_1 + x_2 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_0   1  1256535 588.29
## - x_1   1  1256538 588.29
## - x_3   1  1289895 589.45
## <none>     1242093 589.78
## - x_2   1  1329712 590.78
## - x_4   1  6423684 660.08
```

```
##
## Step:  AIC=588.29
## .outcome ~ x_1 + x_2 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_1   1  1259649 586.40
## - x_3   1  1296587 587.67
## <none>     1256535 588.29
## - x_2   1  1358643 589.73
## - x_4   1  6442314 658.21
##
## Step:  AIC=586.4
## .outcome ~ x_2 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_3   1  1303505 585.91
## <none>     1259649 586.40
## - x_2   1  1396258 588.93
## - x_4   1  6871183 659.05
##
## Step:  AIC=585.91
## .outcome ~ x_2 + x_4
##
##        Df Deviance    AIC
## <none>     1303505 585.91
## - x_2   1  1396258 586.93
## - x_4   1 16966760 696.82
## Start:  AIC=591.28
## .outcome ~ x_0 + x_1 + x_2 + x_3 + x_4
##
##        Df Deviance    AIC
## - x_3   1  1288908 589.41
## - x_2   1  1301176 589.83
## - x_1   1  1305293 589.97
## - x_0   1  1305311 589.97
## <none>     1284924 591.28
## - x_4   1  7371742 666.14
##
## Step:  AIC=589.41
## .outcome ~ x_0 + x_1 + x_2 + x_4
##
##        Df Deviance    AIC
## - x_2   1  1301283 587.83
## - x_1   1  1308434 588.07
## - x_0   1  1308450 588.07
## <none>     1288908 589.41
## - x_4   1 16629018 699.94
##
## Step:  AIC=587.83
## .outcome ~ x_0 + x_1 + x_4
##
##        Df Deviance    AIC
## - x_1   1  1320948 586.49
## - x_0   1  1320962 586.49
```

```
## <none>     1301283 587.83
## - x_4   1 64127448 757.32
##
## Step:  AIC=586.49
## .outcome ~ x_0 + x_4
##
##        Df Deviance    AIC
## - x_0   1  1350072 585.45
## <none>     1320948 586.49
## - x_4   1 74268476 761.78
##
## Step:  AIC=585.45
## .outcome ~ x_4
##
##        Df  Deviance    AIC
## <none>      1350072 585.45
## - x_4   1 463881836 840.39
```

```r
# Make predictions for x_5 in the x_test_set based on the x_model
x_predictions <- predict(x_model, newdata = x_test_set)

# Since I'm not looking to be exact, evaluate the accuracy of the predictions on a sliding scale
glmStepAIC_x_accuracy_within_250_meters <- mean(x_predictions <= x_test_set$x_5 + 250 & x_predictions >=
glmStepAIC_x_accuracy_within_100_meters <- mean(x_predictions <= x_test_set$x_5 + 100 & x_predictions >=
glmStepAIC_x_accuracy_within_50_meters <- mean(x_predictions <= x_test_set$x_5 + 50 & x_predictions >= x
glmStepAIC_x_accuracy_within_25_meters <- mean(x_predictions <= x_test_set$x_5 + 25 & x_predictions >= x

# Train a model to predict the y coordinate of the final stage of the ring in an Apex Legends match
y_model <- train(
  y_5 ~ y_0 + y_1 + y_2 + y_3 + y_4, # Predict on the other 4 y coordinates of the other ring stages
  data = y_train_set,
  method = "glmStepAIC",
  verbose = FALSE # suppress verbose output
)
```

```
## Start:  AIC=594.76
## .outcome ~ y_0 + y_1 + y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_3   1  1397625 592.97
## - y_0   1  1406004 593.24
## - y_1   1  1406030 593.24
## <none>     1390687 594.76
## - y_2   1  1581196 598.40
## - y_4   1  7114527 664.58
##
## Step:  AIC=592.97
## .outcome ~ y_0 + y_1 + y_2 + y_4
##
##        Df Deviance    AIC
## - y_0   1  1409695 591.35
## - y_1   1  1409717 591.35
## <none>     1397625 592.97
## - y_2   1  1596498 596.83
## - y_4   1 17324309 701.74
```

```
##
## Step:  AIC=591.35
## .outcome ~ y_1 + y_2 + y_4
##
##        Df Deviance    AIC
## - y_1   1  1448491 590.55
## <none>     1409695 591.35
## - y_2   1  1619792 595.47
## - y_4   1 18891953 703.55
##
## Step:  AIC=590.55
## .outcome ~ y_2 + y_4
##
##        Df Deviance    AIC
## <none>     1448491 590.55
## - y_2   1  1619803 593.47
## - y_4   1 20283014 704.68
## Start:  AIC=578.47
## .outcome ~ y_0 + y_1 + y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_2   1   962716 576.57
## <none>      960359 578.47
## - y_1   1  1027217 579.43
## - y_0   1  1027249 579.43
## - y_3   1  1063942 580.97
## - y_4   1  6298083 659.22
##
## Step:  AIC=576.57
## .outcome ~ y_0 + y_1 + y_3 + y_4
##
##        Df Deviance    AIC
## <none>      962716 576.57
## - y_1   1  1041107 578.02
## - y_0   1  1041137 578.02
## - y_3   1  1100908 580.47
## - y_4   1  6446363 658.24
## Start:  AIC=572.78
## .outcome ~ y_0 + y_1 + y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_0   1   855548 571.38
## - y_1   1   855551 571.38
## <none>      843939 572.78
## - y_2   1   893366 573.28
## - y_3   1  1372152 592.17
## - y_4   1  7878429 669.07
##
## Step:  AIC=571.38
## .outcome ~ y_1 + y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_1   1   856196 569.41
## <none>      855548 571.38
```

```
## - y_2   1   922568 572.70
## - y_3   1  1381559 590.47
## - y_4   1  7931074 667.36
##
## Step:  AIC=569.41
## .outcome ~ y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## <none>      856196 569.41
## - y_2   1   956012 572.27
## - y_3   1  1432119 590.05
## - y_4   1  8673311 669.30
## Start:  AIC=597.34
## .outcome ~ y_0 + y_1 + y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_1   1  1475069 595.35
## - y_0   1  1475070 595.35
## - y_3   1  1514695 596.51
## - y_2   1  1523759 596.78
## <none>     1474830 597.34
## - y_4   1  4997924 649.04
##
## Step:  AIC=595.35
## .outcome ~ y_0 + y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_0   1  1476412 593.39
## - y_3   1  1515076 594.53
## - y_2   1  1528375 594.91
## <none>     1475069 595.35
## - y_4   1  5012710 647.17
##
## Step:  AIC=593.39
## .outcome ~ y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_3   1  1515566 592.54
## - y_2   1  1540034 593.24
## <none>     1476412 593.39
## - y_4   1  5467209 648.99
##
## Step:  AIC=592.54
## .outcome ~ y_2 + y_4
##
##        Df Deviance    AIC
## - y_2   1  1542153 591.30
## <none>     1515566 592.54
## - y_4   1 25158638 714.15
##
## Step:  AIC=591.3
## .outcome ~ y_4
##
##        Df  Deviance    AIC
```

```
## <none>      1542153 591.30
## - y_4   1 629433843 853.82
## Start:  AIC=605.66
## .outcome ~ y_0 + y_1 + y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_2   1  1784694 603.73
## - y_0   1  1805581 604.24
## - y_1   1  1805598 604.24
## - y_3   1  1813001 604.42
## <none>     1781604 605.66
## - y_4   1  7250726 665.41
##
## Step:  AIC=603.73
## .outcome ~ y_0 + y_1 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_0   1  1818566 602.56
## - y_1   1  1818591 602.56
## - y_3   1  1829344 602.82
## <none>     1784694 603.73
## - y_4   1  7367693 664.12
##
## Step:  AIC=602.56
## .outcome ~ y_1 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_3   1  1860418 601.56
## - y_1   1  1885233 602.14
## <none>     1818566 602.56
## - y_4   1  7369239 662.13
##
## Step:  AIC=601.56
## .outcome ~ y_1 + y_4
##
##        Df Deviance    AIC
## - y_1   1  1929633 601.17
## <none>     1860418 601.56
## - y_4   1 56002907 749.36
##
## Step:  AIC=601.17
## .outcome ~ y_4
##
##        Df  Deviance    AIC
## <none>      1929633 601.17
## - y_4   1 721116187 859.80
## Start:  AIC=589.32
## .outcome ~ y_0 + y_1 + y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_2   1  1230307 587.36
## - y_1   1  1259997 588.41
## - y_0   1  1260018 588.41
## <none>     1229104 589.32
```

```
## - y_3   1  1321570 590.51
## - y_4   1  6608119 661.33
##
## Step:  AIC=587.36
## .outcome ~ y_0 + y_1 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_1   1  1282025 587.18
## - y_0   1  1282047 587.18
## <none>     1230307 587.36
## - y_3   1  1357287 589.69
## - y_4   1  6625436 659.45
##
## Step:  AIC=587.18
## .outcome ~ y_0 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_0   1  1297220 585.69
## <none>     1282025 587.18
## - y_3   1  1393029 588.83
## - y_4   1  6634501 657.51
##
## Step:  AIC=585.69
## .outcome ~ y_3 + y_4
##
##        Df Deviance    AIC
## <none>     1297220 585.69
## - y_3   1  1401106 587.08
## - y_4   1  7050283 658.18
## Start:  AIC=589.9
## .outcome ~ y_0 + y_1 + y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_1   1  1254784 588.23
## - y_0   1  1254797 588.23
## - y_3   1  1257732 588.33
## <none>     1245392 589.90
## - y_2   1  1343291 591.23
## - y_4   1  5480789 653.10
##
## Step:  AIC=588.23
## .outcome ~ y_0 + y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_3   1  1268478 586.71
## - y_0   1  1271691 586.82
## <none>     1254784 588.23
## - y_2   1  1343463 589.24
## - y_4   1  5488335 651.16
##
## Step:  AIC=586.71
## .outcome ~ y_0 + y_2 + y_4
##
##        Df Deviance    AIC
```

```
## - y_0   1  1275441 584.95
## <none>     1268478 586.71
## - y_2   1  1432952 590.07
## - y_4   1 17244898 699.54
##
## Step:  AIC=584.95
## .outcome ~ y_2 + y_4
##
##        Df Deviance    AIC
## <none>     1275441 584.95
## - y_2   1  1450074 588.60
## - y_4   1 20104156 704.29
## Start:  AIC=594.15
## .outcome ~ y_0 + y_1 + y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_2   1  1372327 592.17
## - y_0   1  1409666 593.35
## - y_1   1  1409696 593.35
## <none>     1371658 594.15
## - y_3   1  1437970 594.23
## - y_4   1  5400469 652.45
##
## Step:  AIC=592.17
## .outcome ~ y_0 + y_1 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_0   1  1415635 591.54
## - y_1   1  1415677 591.54
## <none>     1372327 592.17
## - y_3   1  1453033 592.69
## - y_4   1  5449106 650.84
##
## Step:  AIC=591.54
## .outcome ~ y_1 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_1   1  1467541 591.12
## <none>     1415635 591.54
## - y_3   1  1530720 592.98
## - y_4   1  5772057 651.38
##
## Step:  AIC=591.12
## .outcome ~ y_3 + y_4
##
##        Df Deviance    AIC
## <none>     1467541 591.12
## - y_3   1  1548692 591.49
## - y_4   1  6727978 656.12
## Start:  AIC=583.59
## .outcome ~ y_0 + y_1 + y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_2   1  1118214 583.16
```

```
## <none>     1079080 583.59
## - y_0   1  1136198 583.86
## - y_1   1  1136235 583.86
## - y_3   1  1232265 587.43
## - y_4   1  6026808 657.28
##
## Step:  AIC=583.16
## .outcome ~ y_0 + y_1 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_0   1  1150490 582.41
## - y_1   1  1150514 582.41
## <none>     1118214 583.16
## - y_3   1  1234309 585.51
## - y_4   1  6039346 655.37
##
## Step:  AIC=582.41
## .outcome ~ y_1 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_1   1  1184996 581.71
## <none>     1150490 582.41
## - y_3   1  1317215 586.37
## - y_4   1  6588634 657.20
##
## Step:  AIC=581.71
## .outcome ~ y_3 + y_4
##
##        Df Deviance    AIC
## <none>     1184996 581.71
## - y_3   1  1328139 584.73
## - y_4   1  7535868 661.11
## Start:  AIC=595.12
## .outcome ~ y_0 + y_1 + y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_2   1  1403034 593.14
## <none>     1402115 595.12
## - y_1   1  1512596 596.45
## - y_0   1  1512643 596.45
## - y_3   1  1526421 596.85
## - y_4   1  7575550 667.34
##
## Step:  AIC=593.14
## .outcome ~ y_0 + y_1 + y_3 + y_4
##
##        Df Deviance    AIC
## <none>     1403034 593.14
## - y_1   1  1512890 594.46
## - y_0   1  1512933 594.46
## - y_3   1  1548047 595.47
## - y_4   1  7769989 666.46
## Start:  AIC=588.53
## .outcome ~ y_0 + y_1 + y_2 + y_3 + y_4
```

```
##
##        Df Deviance    AIC
## - y_2   1  1210871 586.66
## - y_3   1  1234295 587.51
## <none>     1207225 588.53
## - y_1   1  1298861 589.75
## - y_0   1  1298916 589.75
## - y_4   1  4525695 644.67
##
## Step:  AIC=586.66
## .outcome ~ y_0 + y_1 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_3   1  1236690 585.59
## <none>     1210871 586.66
## - y_1   1  1304159 587.93
## - y_0   1  1304203 587.93
## - y_4   1  4713189 644.46
##
## Step:  AIC=585.59
## .outcome ~ y_0 + y_1 + y_4
##
##        Df Deviance    AIC
## <none>     1236690 585.59
## - y_1   1  1328413 586.74
## - y_0   1  1328450 586.74
## - y_4   1 38606665 735.00
## Start:  AIC=582.5
## .outcome ~ y_0 + y_1 + y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_0   1  1053537 580.54
## - y_1   1  1053539 580.54
## - y_2   1  1056468 580.66
## - y_3   1  1065528 581.04
## <none>     1052530 582.50
## - y_4   1  5793639 655.54
##
## Step:  AIC=580.54
## .outcome ~ y_1 + y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_2   1  1056555 578.67
## - y_1   1  1057470 578.70
## - y_3   1  1065843 579.05
## <none>     1053537 580.54
## - y_4   1  5803371 653.62
##
## Step:  AIC=578.67
## .outcome ~ y_1 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_1   1  1059592 576.79
## - y_3   1  1065954 577.05
```

```
## <none>     1056555 578.67
## - y_4   1  6084266 653.70
##
## Step:  AIC=576.79
## .outcome ~ y_3 + y_4
##
##        Df Deviance    AIC
## - y_3   1  1068262 575.15
## <none>     1059592 576.79
## - y_4   1  6911726 657.31
##
## Step:  AIC=575.15
## .outcome ~ y_4
##
##        Df  Deviance    AIC
## <none>      1068262 575.15
## - y_4   1 513380344 844.85
## Start:  AIC=585.99
## .outcome ~ y_0 + y_1 + y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_1   1  1141457 584.07
## - y_0   1  1141470 584.07
## <none>     1139563 585.99
## - y_2   1  1209825 586.63
## - y_3   1  1231553 587.41
## - y_4   1  7063777 664.26
##
## Step:  AIC=584.07
## .outcome ~ y_0 + y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## <none>     1141457 584.07
## - y_3   1  1232580 585.45
## - y_2   1  1237259 585.61
## - y_0   1  1251087 586.10
## - y_4   1  7089228 662.42
## Start:  AIC=610.36
## .outcome ~ y_0 + y_1 + y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_3   1  1984270 608.40
## - y_0   1  2068115 610.22
## - y_1   1  2068131 610.22
## <none>     1982443 610.36
## - y_2   1  2107056 611.04
## - y_4   1  6093976 657.77
##
## Step:  AIC=608.4
## .outcome ~ y_0 + y_1 + y_2 + y_4
##
##        Df Deviance    AIC
## - y_0   1  2068887 608.23
## - y_1   1  2068900 608.23
```

```
## <none>     1984270 608.40
## - y_2   1  2111024 609.12
## - y_4   1 12901855 688.77
##
## Step:  AIC=608.23
## .outcome ~ y_1 + y_2 + y_4
##
##        Df Deviance    AIC
## - y_1   1  2070155 606.26
## - y_2   1  2160506 608.14
## <none>     2068887 608.23
## - y_4   1 12996157 687.09
##
## Step:  AIC=606.26
## .outcome ~ y_2 + y_4
##
##        Df Deviance    AIC
## - y_2   1  2166403 606.26
## <none>     2070155 606.26
## - y_4   1 13226622 685.86
##
## Step:  AIC=606.26
## .outcome ~ y_4
##
##        Df  Deviance    AIC
## <none>      2166403 606.26
## - y_4   1 766314574 862.47
## Start:  AIC=586.08
## .outcome ~ y_0 + y_1 + y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_0   1  1170798 585.18
## - y_1   1  1170805 585.18
## - y_2   1  1188407 585.84
## <none>     1141709 586.08
## - y_3   1  1567304 598.02
## - y_4   1  7316378 665.81
##
## Step:  AIC=585.18
## .outcome ~ y_1 + y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_1   1  1172742 583.26
## - y_2   1  1193816 584.04
## <none>     1170798 585.18
## - y_3   1  1568930 596.06
## - y_4   1  7334113 663.92
##
## Step:  AIC=583.26
## .outcome ~ y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_2   1  1193825 582.04
## <none>     1172742 583.26
```

```
## - y_3   1  1569923 594.09
## - y_4   1  7762902 664.42
##
## Step:  AIC=582.04
## .outcome ~ y_3 + y_4
##
##        Df Deviance    AIC
## <none>     1193825 582.04
## - y_3   1  1595083 592.79
## - y_4   1  7872732 663.03
## Start:  AIC=604.79
## .outcome ~ y_0 + y_1 + y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_1   1  1746948 602.79
## - y_0   1  1746950 602.79
## <none>     1746849 604.79
## - y_3   1  1876717 605.94
## - y_2   1  1918771 606.92
## - y_4   1  4489871 644.32
##
## Step:  AIC=602.79
## .outcome ~ y_0 + y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_0   1  1772228 601.42
## <none>     1746948 602.79
## - y_3   1  1876752 603.94
## - y_2   1  1957740 605.80
## - y_4   1  4611438 643.50
##
## Step:  AIC=601.42
## .outcome ~ y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## <none>     1772228 601.42
## - y_3   1  1884270 602.12
## - y_2   1  1958578 603.82
## - y_4   1  5151571 646.37
## Start:  AIC=588.64
## .outcome ~ y_0 + y_1 + y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_3   1  1210140 586.64
## <none>     1210140 588.64
## - y_1   1  1402879 593.14
## - y_0   1  1402894 593.14
## - y_2   1  1787563 603.80
## - y_4   1  5599655 654.04
##
## Step:  AIC=586.64
## .outcome ~ y_0 + y_1 + y_2 + y_4
##
##        Df Deviance    AIC
```

```
## <none>     1210140 586.64
## - y_1   1  1406367 591.25
## - y_0   1  1406378 591.25
## - y_2   1  1798041 602.06
## - y_4   1 12779038 688.35
## Start:  AIC=585.31
## .outcome ~ y_0 + y_1 + y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_3   1  1124336 583.40
## - y_0   1  1158738 584.73
## - y_1   1  1158739 584.73
## <none>     1122132 585.31
## - y_2   1  1210097 586.64
## - y_4   1  4193878 641.32
##
## Step:  AIC=583.4
## .outcome ~ y_0 + y_1 + y_2 + y_4
##
##        Df Deviance    AIC
## - y_1   1  1161486 582.83
## - y_0   1  1161488 582.83
## <none>     1124336 583.40
## - y_2   1  1220005 584.99
## - y_4   1 14152574 692.84
##
## Step:  AIC=582.83
## .outcome ~ y_0 + y_2 + y_4
##
##        Df Deviance    AIC
## - y_0   1  1161804 580.84
## <none>     1161486 582.83
## - y_2   1  1250285 584.07
## - y_4   1 14154985 690.85
##
## Step:  AIC=580.84
## .outcome ~ y_2 + y_4
##
##        Df Deviance    AIC
## <none>     1161804 580.84
## - y_2   1  1254561 582.22
## - y_4   1 15060474 691.58
## Start:  AIC=580.76
## .outcome ~ y_0 + y_1 + y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_3   1  1012148 578.78
## - y_1   1  1026066 579.38
## - y_0   1  1026072 579.38
## - y_2   1  1029700 579.53
## <none>     1011753 580.76
## - y_4   1  3914906 638.30
##
## Step:  AIC=578.78
```

```
## .outcome ~ y_0 + y_1 + y_2 + y_4
##
##        Df Deviance    AIC
## - y_1   1  1026433 577.39
## - y_0   1  1026439 577.39
## - y_2   1  1031359 577.60
## <none>     1012148 578.78
## - y_4   1 16765760 700.30
##
## Step:  AIC=577.39
## .outcome ~ y_0 + y_2 + y_4
##
##        Df Deviance    AIC
## - y_0   1  1029151 575.51
## - y_2   1  1039861 575.96
## <none>     1026433 577.39
## - y_4   1 17111646 699.19
##
## Step:  AIC=575.51
## .outcome ~ y_2 + y_4
##
##        Df Deviance    AIC
## - y_2   1  1053563 574.54
## <none>     1029151 575.51
## - y_4   1 17138778 697.26
##
## Step:  AIC=574.54
## .outcome ~ y_4
##
##        Df  Deviance    AIC
## <none>      1053563 574.54
## - y_4   1 715520506 859.46
## Start:  AIC=583.57
## .outcome ~ y_0 + y_1 + y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_0   1  1079689 581.62
## - y_1   1  1079700 581.62
## <none>     1078431 583.57
## - y_3   1  1144676 584.19
## - y_2   1  1162484 584.87
## - y_4   1  6121335 657.96
##
## Step:  AIC=581.62
## .outcome ~ y_1 + y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## <none>     1079689 581.62
## - y_3   1  1158750 582.73
## - y_1   1  1175867 583.37
## - y_2   1  1201001 584.30
## - y_4   1  6169650 656.31
## Start:  AIC=594.26
## .outcome ~ y_0 + y_1 + y_2 + y_3 + y_4
```

```
##
##        Df Deviance    AIC
## - y_3   1  1383532 592.53
## - y_1   1  1396002 592.92
## - y_0   1  1396049 592.93
## - y_2   1  1410604 593.38
## <none>     1374949 594.26
## - y_4   1  4598756 645.38
##
## Step:  AIC=592.53
## .outcome ~ y_0 + y_1 + y_2 + y_4
##
##        Df Deviance    AIC
## - y_1   1  1404148 591.18
## - y_0   1  1404188 591.18
## - y_2   1  1410725 591.39
## <none>     1383532 592.53
## - y_4   1 12229844 686.42
##
## Step:  AIC=591.18
## .outcome ~ y_0 + y_2 + y_4
##
##        Df Deviance    AIC
## <none>     1404148 591.18
## - y_2   1  1476514 591.39
## - y_0   1  1487153 591.71
## - y_4   1 13859652 689.92
## Start:  AIC=583.6
## .outcome ~ y_0 + y_1 + y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_3   1  1083297 581.77
## - y_0   1  1087511 581.94
## - y_1   1  1087519 581.94
## <none>     1079318 583.60
## - y_2   1  1192945 586.01
## - y_4   1  5986865 656.99
##
## Step:  AIC=581.77
## .outcome ~ y_0 + y_1 + y_2 + y_4
##
##        Df Deviance    AIC
## - y_0   1  1094730 580.23
## - y_1   1  1094740 580.23
## <none>     1083297 581.77
## - y_2   1  1232869 585.46
## - y_4   1 12694839 688.06
##
## Step:  AIC=580.23
## .outcome ~ y_1 + y_2 + y_4
##
##        Df Deviance    AIC
## - y_1   1  1108726 578.79
## <none>     1094730 580.23
```

```
## - y_2   1  1235683 583.56
## - y_4   1 13697942 689.40
##
## Step:  AIC=578.79
## .outcome ~ y_2 + y_4
##
##        Df Deviance    AIC
## <none>     1108726 578.79
## - y_2   1  1238008 581.64
## - y_4   1 14970401 691.31
## Start:  AIC=593.43
## .outcome ~ y_0 + y_1 + y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_2   1  1389932 592.73
## <none>     1349388 593.43
## - y_0   1  1461273 594.93
## - y_1   1  1461338 594.94
## - y_3   1  1536827 597.15
## - y_4   1  7037400 664.10
##
## Step:  AIC=592.73
## .outcome ~ y_0 + y_1 + y_3 + y_4
##
##        Df Deviance    AIC
## <none>     1389932 592.73
## - y_0   1  1461570 592.94
## - y_1   1  1461631 592.95
## - y_3   1  1537323 595.17
## - y_4   1  7329362 663.89
## Start:  AIC=610.07
## .outcome ~ y_0 + y_1 + y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_3   1  1969879 608.08
## - y_2   1  1970487 608.09
## - y_0   1  1970573 608.09
## - y_1   1  1970574 608.09
## <none>     1969635 610.07
## - y_4   1  6168487 658.30
##
## Step:  AIC=608.08
## .outcome ~ y_0 + y_1 + y_2 + y_4
##
##        Df Deviance    AIC
## - y_0   1  1970813 606.10
## - y_1   1  1970814 606.10
## - y_2   1  1971419 606.11
## <none>     1969879 608.08
## - y_4   1 14762377 694.70
##
## Step:  AIC=606.1
## .outcome ~ y_1 + y_2 + y_4
##
```

```
##        Df Deviance    AIC
## - y_1   1  1970987 604.10
## - y_2   1  1973888 604.16
## <none>     1970813 606.10
## - y_4   1 15682064 695.36
##
## Step:  AIC=604.1
## .outcome ~ y_2 + y_4
##
##        Df Deviance    AIC
## - y_2   1  1974783 602.18
## <none>     1970987 604.10
## - y_4   1 16966101 696.82
##
## Step:  AIC=602.18
## .outcome ~ y_4
##
##        Df  Deviance    AIC
## <none>      1974783 602.18
## - y_4   1 675417092 856.92
## Start:  AIC=572.48
## .outcome ~ y_0 + y_1 + y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## <none>      838261 572.48
## - y_2   1   886662 572.95
## - y_0   1   954426 576.19
## - y_1   1   954442 576.19
## - y_3   1   978564 577.29
## - y_4   1  6065060 657.56
## Start:  AIC=596.89
## .outcome ~ y_0 + y_1 + y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_1   1  1460178 594.90
## - y_0   1  1460178 594.90
## - y_3   1  1479577 595.48
## - y_2   1  1481464 595.54
## <none>     1459708 596.89
## - y_4   1  6873897 663.06
##
## Step:  AIC=594.9
## .outcome ~ y_0 + y_2 + y_3 + y_4
##
##        Df Deviance    AIC
## - y_0   1  1460271 592.90
## - y_3   1  1479607 593.48
## - y_2   1  1489064 593.76
## <none>     1460178 594.90
## - y_4   1  6909593 661.29
##
## Step:  AIC=592.9
## .outcome ~ y_2 + y_3 + y_4
##
```

```
##        Df Deviance    AIC
## - y_3   1  1481244 591.53
## - y_2   1  1491762 591.84
## <none>     1460271 592.90
## - y_4   1  7556584 663.23
##
## Step:  AIC=591.53
## .outcome ~ y_2 + y_4
##
##        Df Deviance    AIC
## <none>     1481244 591.53
## - y_2   1  1551135 591.56
## - y_4   1 18288039 700.12
```

```r
# Make predictions for y_5 in the y_test_set based on the y_model
y_predictions <- predict(y_model, newdata = y_test_set)

# Since I'm not looking to be exact, evaluate the accuracy of the predictions on a sliding scale
glmStepAIC_y_accuracy_within_250_meters <- mean(y_predictions <= y_test_set$y_5 + 250 & y_predictions >=
glmStepAIC_y_accuracy_within_100_meters <- mean(y_predictions <= y_test_set$y_5 + 100 & y_predictions >=
glmStepAIC_y_accuracy_within_50_meters <- mean(y_predictions <= y_test_set$y_5 + 50 & y_predictions >= y
glmStepAIC_y_accuracy_within_25_meters <- mean(y_predictions <= y_test_set$y_5 + 25 & y_predictions >= y

# Print and save the accuracy results to a results variable
accuracy_results <- bind_rows(
  accuracy_results,
  data.frame(
    method = "glmStepAIC",
    x_250 = glmStepAIC_x_accuracy_within_250_meters,
    x_100 = glmStepAIC_x_accuracy_within_100_meters,
    x_50 = glmStepAIC_x_accuracy_within_50_meters,
    x_25 = glmStepAIC_x_accuracy_within_25_meters,
    y_250 = glmStepAIC_y_accuracy_within_250_meters,
    y_100 = glmStepAIC_y_accuracy_within_100_meters,
    y_50 = glmStepAIC_y_accuracy_within_50_meters,
    y_25 = glmStepAIC_y_accuracy_within_25_meters
  )
)
accuracy_results %>% knitr::kable()
```

method	x_250	x_100	x_50	x_25	y_250	y_100	y_50	y_25
glm	0.8750000	0.3958333	0.2083333	0.1666667	0.7708333	0.2916667	0.0833333	0.0416667
glmStepAIC	0.8541667	0.4375000	0.2708333	0.1666667	0.7708333	0.2916667	0.1458333	0.0833333

```r
# Save the results to a RData file
save(accuracy_results, file = "rda/accuracy_results.rda")

rmse_results <- bind_rows(
  rmse_results,
  data.frame(
    method = "glmStepAIC",
    x_RMSE = x_model$results$RMSE,
    y_RMSE = y_model$results$RMSE
```

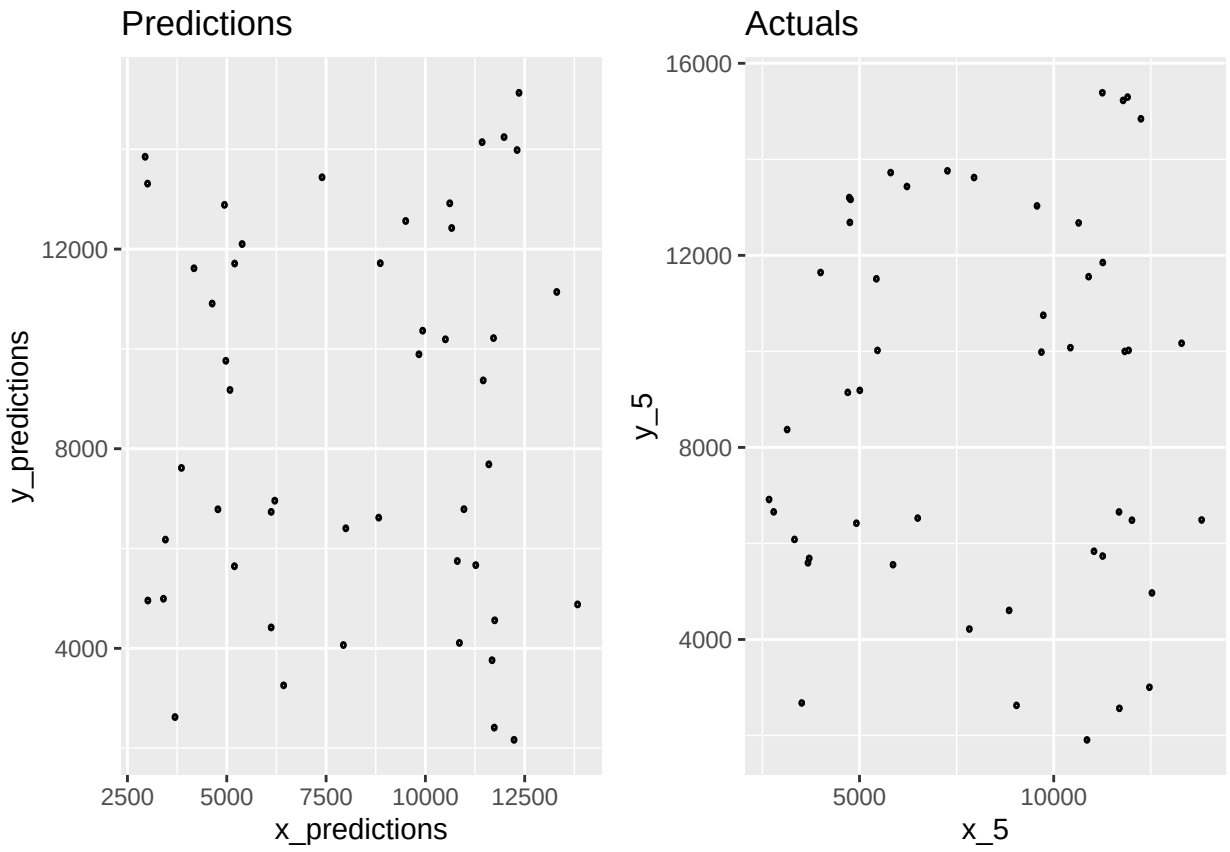
```
  )
)
rmse_results %>% knitr::kable()
```

method	x_RMSE	y_RMSE
glm	222.6889	230.7225
glmStepAIC	192.4899	226.0049

```
# Save the results to a RData file
save(rmse_results, file = "rda/rmse_results.rda")
```

The rest of the R code is similar to the "Generalized Linear Model" code, where I gather the accuracy and and RMSEs of each model's predictions. Below is the plot displaying the predicted final ring locations against the actual final ring locations.

```
# Plot a circle the size of the final ring for each x and y coordinate prediction
predictions_plot <- data.frame(x_predictions, y_predictions) %>%
  ggplot(aes(x0 = x_predictions, y0 = y_predictions, r = ring_5_diameter / 2)) +
  geom_circle() +
  ggtitle("Predictions")
# Plot a circle the size of the final ring for each actual x and y coordinate
actuals_plot <- apex_data_wide %>%
  filter(gameID %in% x_test_set$gameID) %>%
  ggplot(aes(x0 = x_5, y0 = y_5, r = ring_5_diameter / 2)) +
  geom_circle() +
  ggtitle("Actuals")
# Display the two plots side-by-side
grid.arrange(predictions_plot, actuals_plot, ncol = 2)
```



3.4 Least Angle Regression

After testing a large number of algorithms made available through the `caret`(Kuhn and Max 2008), I decided on using the "Least Angle Regression" method, or `lars` for short. You can see more information about this method as well by executing the R code shown below.

```
getModelInfo("lars")
```

This is a different type of regression from the linear regression we are used to seeing. Here, the regression will find which predictor has the highest correlation with the target and tries to fit a model that will plot an angle most similarly to that of the angle made by the plot of the target. I chose this algorithm out of all the others that I tested because

1. The RMSE most closely resembled those that have already been shown in this capstone, and
2. When viewing the various stages of the ring in an Apex Legends match, you could capture the distance and angle of the trajectory of the ring by drawing lines in between the center points of the ring location at its different stages.

```
# Least Angle Regression
# Train a model to predict the x coordinate of the final stage of the ring in an Apex Legends match
x_model <- train(
  x_5 ~ x_0 + x_1 + x_2 + x_3 + x_4, # Predict on the other 4 x coordinates of the other ring stages
  data = x_train_set,
  method = "lars"
)

# Make predictions for x_5 in the x_test_set based on the x_model
x_predictions <- predict(x_model, newdata = x_test_set)
```

```r
# Since I'm not looking to be exact, evaluate the accuracy of the predictions on a sliding scale
lars_x_accuracy_within_250_meters <- mean(x_predictions <= x_test_set$x_5 + 250 & x_predictions >= x_tes
lars_x_accuracy_within_100_meters <- mean(x_predictions <= x_test_set$x_5 + 100 & x_predictions >= x_tes
lars_x_accuracy_within_50_meters <- mean(x_predictions <= x_test_set$x_5 + 50 & x_predictions >= x_test_
lars_x_accuracy_within_25_meters <- mean(x_predictions <= x_test_set$x_5 + 25 & x_predictions >= x_test_

# Train a model to predict the y coordinate of the final stage of the ring in an Apex Legends match
y_model <- train(
  y_5 ~ y_0 + y_1 + y_2 + y_3 + y_4, # Predict on the other 4 y coordinates of the other ring stages
  data = y_train_set,
  method = "lars"
)

# Make predictions for y_5 in the y_test_set based on the y_model
y_predictions <- predict(y_model, newdata = y_test_set)

# Since I'm not looking to be exact, evaluate the accuracy of the predictions on a sliding scale
lars_y_accuracy_within_250_meters <- mean(y_predictions <= y_test_set$y_5 + 250 & y_predictions >= y_tes
lars_y_accuracy_within_100_meters <- mean(y_predictions <= y_test_set$y_5 + 100 & y_predictions >= y_tes
lars_y_accuracy_within_50_meters <- mean(y_predictions <= y_test_set$y_5 + 50 & y_predictions >= y_test_
lars_y_accuracy_within_25_meters <- mean(y_predictions <= y_test_set$y_5 + 25 & y_predictions >= y_test_

# Print and save the accuracy results to a results variable
accuracy_results <- bind_rows(
  accuracy_results,
  data.frame(
    method = "lars",
    x_250 = lars_x_accuracy_within_250_meters,
    x_100 = lars_x_accuracy_within_100_meters,
    x_50 = lars_x_accuracy_within_50_meters,
    x_25 = lars_x_accuracy_within_25_meters,
    y_250 = lars_y_accuracy_within_250_meters,
    y_100 = lars_y_accuracy_within_100_meters,
    y_50 = lars_y_accuracy_within_50_meters,
    y_25 = lars_y_accuracy_within_25_meters
  )
)
accuracy_results %>% knitr::kable()
```

method	x_250	x_100	x_50	x_25	y_250	y_100	y_50	y_25
glm	0.8750000	0.3958333	0.2083333	0.1666667	0.7708333	0.2916667	0.0833333	0.0416667
glmStepAIC	0.8541667	0.4375000	0.2708333	0.1666667	0.7708333	0.2916667	0.1458333	0.0833333
lars	0.8750000	0.3958333	0.2083333	0.1666667	0.7708333	0.2916667	0.0833333	0.0416667

```r
# Save the results to a RData file
save(accuracy_results, file = "rda/accuracy_results.rda")

rmse_results <- bind_rows(
  rmse_results,
  data.frame(
    method = "lars",
    x_RMSE = x_model$results$RMSE,
```

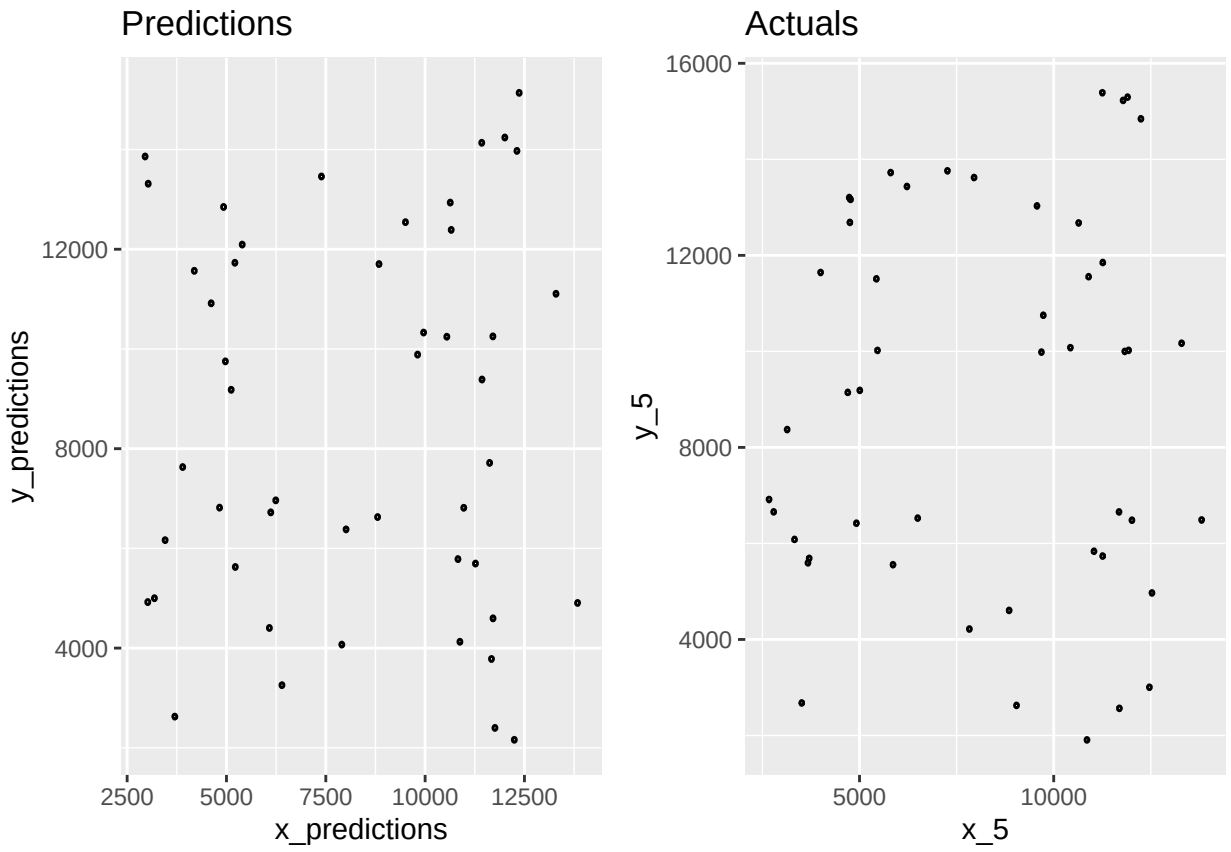
```
    y_RMSE = y_model$results$RMSE
  )
)
rmse_results %>% knitr::kable()
```

method	x_RMSE	y_RMSE
glm	222.6889	230.7225
glmStepAIC	192.4899	226.0049
lars	883.9076	484.6996
lars	441.3654	213.7258
lars	201.7753	217.0676

```
# Save the results to a RData file
save(rmse_results, file = "rda/rmse_results.rda")
```

Again, the rest of the R code should be familiar, where I gather the accuracy and and RMSEs of each model's predictions. Also, below is the usual plot displaying the predicted final ring locations against the actual final ring locations.

```
# Plot a circle the size of the final ring for each x and y coordinate prediction
predictions_plot <- data.frame(x_predictions, y_predictions) %>%
  ggplot(aes(x0 = x_predictions, y0 = y_predictions, r = ring_5_diameter / 2)) +
  geom_circle() +
  ggtitle("Predictions")
# Plot a circle the size of the final ring for each actual x and y coordinate
actuals_plot <- apex_data_wide %>%
  filter(gameID %in% x_test_set$gameID) %>%
  ggplot(aes(x0 = x_5, y0 = y_5, r = ring_5_diameter / 2)) +
  geom_circle() +
  ggtitle("Actuals")
# Display the two plots side-by-side
grid.arrange(predictions_plot, actuals_plot, ncol = 2)
```



4 Conclusion

After all the iterations on the methods used to train the algorithm, here are where the accuracy of each model stands.

method	x_250	x_100	x_50	x_25	y_250	y_100	y_50	y_25
glm	0.8750000	0.3958333	0.2083333	0.1666667	0.7708333	0.2916667	0.0833333	0.0416667
glmStepAIC	0.8541667	0.4375000	0.2708333	0.1666667	0.7708333	0.2916667	0.1458333	0.0833333
lars	0.8750000	0.3958333	0.2083333	0.1666667	0.7708333	0.2916667	0.0833333	0.0416667

Also, here are where the RMSE results for each method stand.

method	x_RMSE	y_RMSE
glm	222.6889	230.7225
glmStepAIC	192.4899	226.0049
lars	883.9076	484.6996
lars	441.3654	213.7258
lars	201.7753	217.0676

As we can see, the “Generalized Linear Model With Stepwise Feature Selection” had the best RMSE for the x coordinate, an iteration of the “Least Angle Regression” had the best RMSE for the y coordinate, and both the “Generalized Linear Model” and “Least Angle Regression” had similar scores when it came to accuracy

across the board. This indicates to me that, in the future, an ensemble algorithm, where multiple algorithms work together, would best suit a model for predicting the final ring position in an Apex Legends match, and that the same algorithm is not necessarily the best fit for the x and the y coordinates of said ring position.

4.1 Future Considerations

In the future, I would be interested to see what other features I can add to my data sets and see if they affect my model at all. I would also love to conduct this research on the other battle royale maps in Apex Legends. Additionally, it would be incredible to put my model to the test in a live Apex Legends battle royale match. Lastly, I plan on sharing this experiment with my brother to see what he has to say, since he's much better at the game and been playing Apex Legends for much longer than I have.

4.2 Final Remarks

Thanks to all who read through this report and look through my repository. I look forward to applying my newly gained machine learning knowledge on future projects, and hope to continue expanding that knowledge in this ever-growing field.

References

"Apex Legends - About." 2019. https://www.ea.com/games/apex-legends/about.

"Apex Legends - Battle Royale." 2019. https://www.ea.com/games/apex-legends/modes/battle-royale.

"Apex Legends - Battle Royale - Maps." 2019. https://www.ea.com/games/apex-legends/maps#battle-royale.

"Apex Legends - Characters." 2019. https://www.ea.com/games/apex-legends/about/characters.

"Apex Legends - Endzone & Ring Prediction." 2022. https://github.com/ccamfpsApex/ApexLegendsGuide/wiki/Endzone-&-Ring-Prediction.

"Apex Zone Predict Machine Learning." 2023. https://github.com/bluelightgit/apex-zone-predict-machine-learning/tree/main.

Auguie, Baptiste. 2017. *gridExtra: Miscellaneous Functions for "Grid" Graphics*. https://CRAN.R-project.org/package=gridExtra.

Kuhn, and Max. 2008. "Building Predictive Models in r Using the Caret Package." *Journal of Statistical Software* 28 (5): 1–26. https://doi.org/10.18637/jss.v028.i05.

Ooms, Jeroen. 2014. "The Jsonlite Package: A Practical and Consistent Mapping Between JSON Data and r Objects." *arXiv:1403.2805 [Stat.CO]*. https://arxiv.org/abs/1403.2805.

Pedersen, Thomas Lin. 2022. *Ggforce: Accelerating 'Ggplot2'*. https://CRAN.R-project.org/package=ggforce.

Pedersen, Thomas Lin, and David Robinson. 2022. *Gganimate: A Grammar of Animated Graphics*. https://CRAN.R-project.org/package=gganimate.

Wickham, Hadley. 2016. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. https://ggplot2.tidyverse.org.

Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemun, et al. 2019. "Welcome to the tidyverse." *Journal of Open Source Software* 4 (43): 1686. https://doi.org/10.21105/joss.01686.

Yu, Guangchuang. 2023. *Ggimage: Use Image in 'Ggplot2'*. https://CRAN.R-project.org/package=ggimage.